GEOMETRIC APPROACHES TO DISCRIMINATIVE TRAINING

by

Yuan Cao

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

June, 2015

Abstract

Discriminative training as a general machine learning approach has wide applications in tasks like Natural Language Processing (NLP) and Automatic Speech Recognition (ASR). In this thesis, we are interested in online methods for discriminative training due to their simplicity, efficiency and scalabililty. The novel methods we propose are summarized as follows.

First, an interesting subclass of online learning algorithms adopts multiplicative instead of additive strategies to update parameters of linear models, but none of them can be directly used for structured prediction as required by many NLP tasks. We extend the multiplicative Winnow algorithm to a structured version, and the additive MIRA algorithm to a multiplicative version, and apply the them to NLP tasks. We also give interpretations to the relationship between EG and `prod`, two multiplicative algorithms, from an information geometric perspective.

Secondly, although general online learning frameworks, notably the Online Mirror Descent (OMD), exist and subsume many specific algorithms, they are not suitable for deriving multiplicative algorithms. We therefore propose a new general framework named

ABSTRACT

Generalized Multiplicative Update (GMU) that is multiplicative in nature and easily derives many specific multiplicative algorithms. We then propose a subclass of GMU, named the q-Exponentiated Gradient (qEG) method, that elucidates the relationship among several of the algorithms. To better understand the difference between OMD and GMU, we give further analysis of these algorithms from a Riemannian geometric perspective. We also extend OMD and GMU to accelerated versions by adding momentum terms.

Thirdly, although natural gradient descent (NGD) is often hard to be applied in practice due its computational difficulty, we propose a novel approach for CRF training which allows efficient application of NGD. The loss functions, defined by Bregman divergence, generalizes the log-likelihood objective and can be easily coupled with NGD for optimization. The proposed framework is flexible, allowing us to choose proper convex functions that leads to better training performance.

Finally, traditional vector space linear models require estimating as many parameters as the number of model features. In the presence of millions of features, a common phenomenon in many NLP tasks, this may complicate the training procedure especially when labeled training data is scarce. We propose a novel online learning approach by shifting from vector space to tensor space, which dramatically reduces the number of parameters to be estimated. The resulting model is highly regularized and is particularly suitable for training in low-resource environments.

Primary Reader: Sanjeev Khudanpur

Secondary Reader: Andreas Andreou

ABSTRACT

Third Reader: Trac D. Tran

Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Sanjeev Khudanpur, for his constant support, encouragement and guidance through the years. Prof. Sanjeev has a broad interest in many problems and rich experience in research. He has been very patient with my progress and tolerant with all sorts of mistakes I made, kept encouraging me to pursue my research interests, and coached me how to do error analysis and improve ideas when some approach turns out not to be working. My research skills improved significantly during this process, and I felt lucky to have been given much freedom to explore topics and ideas that I found interesting. I would also like to thank Prof. Andreas Andreou and Prof. Trac. D. Tran for reading my thesis and providing valuable feedback.

I am grateful to have been a part of CLSP, one of the best research labs in natural language processing and speech recognition. I was also privileged to get to know and collaborate with many brilliant and interesting people from the lab: Puyang Xu, Ming Sun, Guoguo Chen, Xuchen Yao, Chunxi Liu, Xiaohui (Samuel) Zhang, Hainan Xu, Nanyun Peng, Mo Yu, Gaurav Kumar, Matt Post, Adam Lopez, Minghua Wu, Adi Renduchintala. Thank you all for your help through the years, and the PhD life would have been much more

boring without your accompany. I was also fortunate to have collaborated with two nice hosts of my internships during the third and fourth year: Abe Ittycheriah at IBM Watson Research Center and Klaus Macherey at Google Translate, with whom I had good times and rewarding experiences.

Finally, I am deeply grateful to my family for their consistent support, love, and encouragement since the beginning of this long journey, without which the completion of the program would have been impossible.

Contents

List of Tables

LIST OF TABLES

List of Figures

Chapter 1

Introduction

1.1 Generative and Discriminative Training

Machine learning algorithms can typically be divided into two major categories: generative and discriminative. The generative approach aims to estimate a distribution over all variables (inputs and outputs) in a system, whereas the discriminative approach adopts a minimalist philosophy and only attempts to optimize a mapping from inputs to the desired outputs, saving all the unnecessary modeling efforts. A generative model is typically trained via maximum likelihood estimation (MLE), whereas a discriminative model is typically trained via empirical risk minimization (ERM).

As two fundamentally different philosophies of machine learning, they have lead to a lot of discussions about the advantages and disadvantages of each approach 1–6. In order to get the best out of both sides, efforts have also been made to merge the two philosophies 7–10.

To give a concrete example, consider a statistical machine translation (MT) task 11. The input x is a German sentence and output y is a candidate English translation. If a generative model is used, the problem of translation will be to find $y = \arg\max_{y'} p(x, y') = \arg\max_{y'} p(x|y')p(y')$. Here $p(y')$ is the language model and $p(x|y')$ can often be decomposed as $p(x|y') = \prod_{i=1}^{L} t(x_i|y'_i)d(start_i - end_{i-1})$, where $t(x_i|y'_i)$ is the phrase translation model and $d(start_i - end_{i-1})$ is the distortion model. It can be seen that in this case only three features are used ($p(y')$, $t(x_i|y'_i)$, d), since the joint probability has to be properly decomposed. However if we shift to a discriminative model, a log-linear model can be used to represent $p(x|y') \propto \exp\{\theta_1 t(x|y') + \theta_2 d(start, end) + \theta_3 p(y')\}$, and we may add other arbitrary features to the model. Now the learning problem becomes finding a set of parameters $\boldsymbol{\theta}$ so that the prediction given by the model $y = \arg\max_{y'} \boldsymbol{\theta}^T \boldsymbol{\Phi}(x, y')$ is as accurate as possible, where $\boldsymbol{\Phi}(x, y')$ is the feature vector.

We briefly summarize the pros and cons of the two approaches in Table 1.1.

Table 1.1: Pros and cons of generative and discriminative approaches.

Approach	Pros	Cons
Generative	Easy to incorporate prior knowledge	Sensitive to modeling error
	Parameter estimation relatively easy	Hard to add rich feature sets
	Less prone to over-fitting	High bias
Discriminative	Focus on a specific task	Good loss function often intractable
	Easy to add rich feature sets	Parameter estimation difficult
	Low model complexity	More prone to over-fitting
	Robust to modeling error	Low bias

In this thesis, we focus on the discriminative training scheme and its applications to natural language processing (NLP) problems, due to its superior empirical performance

illustrated in many cases.

1.2 Problem Statement

In a nutshell, discriminative training is composed of three steps:

1. Model construction:

In order to predict an output y given an input x, we need to first build a parameterized model that captures the relationship between x and y, for example the conditional distribution $p(y|x)$which is often represented in log-linear form. [1]. Usually human knowledge in a specific task is incorporated into the model. A commonly used model for structured prediction problems (like many NLP tasks) is the probabilistic graphical model, which enables flexible representation of causal or dependency relationships among multiple factors. A typical example is using Conditional Random Fields (CRF) for Part-of-Speech (POS) tagging. When the structure of the problem is simple, we may just add manually picked features to the log-linear formulation without graphical modeling. The MT example given in Section 1.1 is such a case. The probabilistic model, thanks to the log-linear formulation, can be simplified to a linear model during decoding.

2. Training objective definition:

Having constructed the model, the next thing to do is to define a training criterion so that the model parameters can be properly estimated. In contrast to generative learning for

[1]For discriminative learning, the model does not have to be a conditional probabilistic model. In fact, it can be any function $y = f(x)$ that sets up a map between x and y. For example, a simple case is to use a vector-space linear model $y = \vec{\theta} \cdot \vec{x}$.

which the goal of parameter estimation is to maximize the joint probability of the training data, which often permits closed-form solutions (for example the Baum-Welch algorithm for HMM), discriminative training requires the training objective to be task-oriented and often forbids closed-form solutions. Obviously, the most direct objective would be to optimize the task-specific performance metric (for example the BLEU score 12 for MT systems). However such objective functions are usually non-smooth and non-convex, posing a big challenge to the optimization algorithm (although such methods do exist, see Section 1.4.4 for an example). Therefore, a common practice is to choose convex surrogate loss functions like hinge-loss, log-loss or exp-loss , which upper-bounds the direct loss, as the training objective. On top of the loss function itself, regularization terms (say l_1 or l_2 norm of the parameter vector) are often added to the objective.

3. Parameter estimation:

Once the training criterion has been defined, all that is left to do is to optimize the objective function given labeled training data. In many discriminative training problems, a key requirement for the optimization algorithm is scalability and efficiency. This is because since we are allowed to add arbitrary features to a discriminative model, the number of features can quickly grow to the order of thousands or even millions (for example the tri-gram features added to a discriminative language model). Therefore, the optimization algorithm is expected to handle training readily in the presence of high-dimensional parameter vectors and large amounts of training data. To this end, online learning algorithms are often chosen for large-scale discriminative training due to their simplicity and scalability.

A further desired property of the optimization algorithm is that they yield sparse solutions, since although millions of discriminative features are defined, many of them may not be very informative and should be assigned weights close or equal to zero.

The problem of discriminative training can be now summarized as follows: Given an input $x \in X$, where X is the input domain of a task, a decoder $\mathbf{GEN}(x) : X \to Y$ generates a set of candidate outputs in the output domain Y based on the current model. The goal of discriminative training is to train the model parameterized by $\boldsymbol{\theta} \in \mathbb{R}^d$ so that the predictions given by a scoring function

$$y^0 = \underset{\hat{y} \in \mathbf{GEN}(x)}{\operatorname{argmax}} f(x, \hat{y}; \boldsymbol{\theta}) \tag{1.1}$$

incur as little loss as possible, according to some task-specific quality measure. A typical scoring function has linear form $f(x, y; \boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{\Phi}(x, y)$, where $\boldsymbol{\Phi}(x, y) \in \mathbb{R}^d$ is the feature vector extracted from the pair (x, y). An illustration of discriminative training is given in Figure 1.1.

This thesis focuses on developing techniques for steps 2 and 3 above, assuming the underlying model is given. We are mainly interested in online learning algorithms for reasons given above. We provide analysis and develop novel multiplicative online learning algorithms based on a new general framework inspired by the study of information geometry (Chapter 3 4), as well as learning algorithms suitable for low-resource environments (Chapter 6). We also define novel training objectives that enables efficient parameter estimation

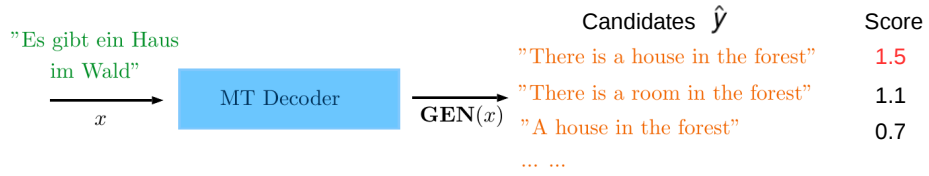


Figure 1.1: An example of discriminative training: In this MT pipeline, given an input sentence x in German, the MT decoder generates a set of candidate translations $\mathbf{GEN}(x)$ in English. Some translations (say "There is a house in the forest") are more correct than others (say "A house in the forest"), and we would like to train the model so that good outputs can be discriminated from bad ones, according to some performance metric.

for CRF models.

1.3 Discriminative Training for NLP and ASR

There has been a long history of applying discriminative training techniques to solving NLP problems. Early examples include maximum entropy modeling for language modeling, word reordering, POS tagging etc. 13–15. In 2002, Michael Collins extended the classical Perceptron classifier 16 to a structured version 17 which reshaped the landscape of discriminative training in NLP. It was also around that time, discriminative structured prediction algorithms from the machine learning community came to be adopted 18. Following this trend, many theoretically-sound prediction algorithms were developed in later years 19 and introduced to the NLP community, making discriminative training a more mature area and an indispensable component of many NLP techniques.

We list some of the most popular discriminative training algorithms applied to NLP tasks, and roughly divide them into online- and batch-learning categories:

- Online learning: (structured) Perceptron 17,20,21, Passive-aggressive/MIRA 22–25, AROW 26, 27, AdaGrad 28 etc.
- Batch learning: MERT 29, PRO 30, Rampion 31, M3N 18, structured SVM 32, Min-Risk 33, 34 etc.

These discriminative training techniques have been applied to almost all kinds of NLP tasks. For example, parsing 35–38, language modeling 39–42, machine translation 24, 43–46, word alignment 47–49, POS tagging 50–53, noun-phrase chunking 54 etc.

For automatic speech recognition (ASR), lots of interesting discriminative training approaches have also been developed and widely used for acoustic and language modeling. The most frequently used methods include MPE, MCE, MMI, bMMI. Generally speaking, these methods define discriminative loss functions on different levels of ASR outputs (MPE for sub-utterance sequence, MCE for utterance sequence and MMI for super-utterance sequence), which are then minimized to find optimal acoustic model parameters. We do not elaborate on this topic here, but good reviews can be found, for example, in 55–57.

1.4 Examles of Discriminative Training Algorithms

1.4.1 Perceptron

Perceptron is perhaps the simplest algorithm for discriminative training. Unlike the original Perceptron 16 proposed as a binary classifier, the widely used Perceptron for many

structured prediction problems in NLP is the structured Perceptron 17. Upon receiving an input x_t at time t, the current model makes a prediction y' according to Eq. 1.1. Whenever the prediction is incorrect ($y' \neq y_t$ where $y_t \in \mathbf{GEN}(x_t)$ is the oracle label for x_t), the algorithm updates the parameters in the following way:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \lambda \left(\boldsymbol{\Phi}(x_t, y_t) - \boldsymbol{\Phi}(x_t, y')\right)$$

where λ is the learning rate.

It can be shown that when the data is separable, namely there exists a $\boldsymbol{\theta}^*$ and $\delta > 0$ such that $\boldsymbol{\theta}^{*T}\boldsymbol{\Phi}(x_t, y_t) \geq \boldsymbol{\theta}^{*T}\boldsymbol{\Phi}(x_t, y') + \delta$, $\forall i$, $\forall y \in \overline{\mathbf{GEN}}(x_t)$, $\mathbf{GEN}(x_t) - \{y_t\}$, then Perceptron is guaranteed to converge while also making less than $\frac{R^2 \|\boldsymbol{\theta}^*\|_2^2}{\delta^2}$ mistakes, where $R = \max_{i, y' \in \overline{\mathbf{GEN}}(x_t)} \|\boldsymbol{\Phi}(x_t, y_t) - \boldsymbol{\Phi}(x_t, y')\|_2$. Although this separability assumption is rarely satisfied in real situations, the Perceptron still performs well in practice and is highly scalable due to its simplicity. Perceptron can be treated as stochastic gradient descent, or a special instance of online mirror descent (OMD) (see Section 2.1) where hinge-loss with zero margin is used as the objective. The lack of margin (hence weak generalization ability) is a major problem with the algorithm.

1.4.2 MIRA

The Margin-Infused Relaxed Algorithm (MIRA) 22, 23 [2] can be treated as an online version of SVM, or a large-margin version of Perceptron. Upon receiving the t^{th} training example, MIRA updates the weight vector by solving the problem

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \frac{1}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|^2 + C\xi$$
$$s.t. \quad \grave{}(\boldsymbol{\theta}, (x_t, y_t)) \leq \xi, \xi \geq 0$$

where $\grave{}(\boldsymbol{\theta}, (x_t, y_t)) \equiv \boldsymbol{\theta} \cdot \boldsymbol{\Phi}(x_t, y') - \boldsymbol{\theta} \cdot \boldsymbol{\Phi}(x_t, y_t)$ is named the structured hinge-loss, in which $y' = \operatorname{argmax}_{y \in \mathbf{GEN}(x_t)} (\boldsymbol{\theta}_t \cdot \boldsymbol{\Phi}(x_t, y) + \rho(y, y_t))$ is the prediction considering the cost $\rho(y, y_t)$[3], which is the difference of the task-specific performance measure between y' and y_t. C is a constant hyper-parameter that penalizes the violation of the margin requirement. This problem has a solution of the form

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \lambda_t \Delta\boldsymbol{\Phi}_t, \text{ with}$$
$$\lambda_t = \min\left(C, \frac{\rho_t - \boldsymbol{\theta}_t \cdot \Delta\boldsymbol{\Phi}_t}{\|\Delta\boldsymbol{\Phi}_t\|^2}\right),$$

[2] Actually the algorithm introduced in this section is named Passive-Aggressive (PA) online learining proposed in 23, and the MIRA was proposed in the author's earlier paper 22. However in NLP literature, MIRA is usually referred as the PA algorithm (for exampe 25, 58). We therefore follow this convention and use the name MIRA here.

[3] The inference procedure to find y' is called "cost-augmented" decoding, in contrast to the commonly used Viterbi decoding in which the cost is not taken into consideration. Hence, to allow dynamic programming, cost-augmented decoding requires the cost function to be decomposable so that it is (approximately) the sum of "local" cost functions.

where $\Delta\boldsymbol{\Phi}_t = \boldsymbol{\Phi}(x_t, y_t) - \boldsymbol{\Phi}(x_t, y')$. It can be seen that compared with the Perceptron, the MIRA update is very similar except that an adaptive learning rate is specified in order to satisfy the large-margin requirement. On the other hand, MIRA can also be treated as a special case of OMD with hinge-loss being the objective and an adaptive learning rate.

In 23, it is shown that MIRA gives a cumulative cost upper-bound as follows:

$$\sum_{t=1}^{T} \rho_t \leq \|\boldsymbol{\theta}\|_2^2 R_2^2 + 2CR_2^2 \sum_{t=1}^{T} \ell_t(\boldsymbol{\theta})$$

where $\|\boldsymbol{\theta}\|$ is an arbitrary weight vector in $\mathbb{R}^d$, $\ell_t(\boldsymbol{\theta})$ be the maximum loss at round t given a weight vector $\boldsymbol{\theta}$, and we assume that $\|\boldsymbol{\Phi}(x_t, y_t) - \boldsymbol{\Phi}(x_t, y)\|_2 \leq R_2$, $\forall t$, $\forall y \in \overline{\mathbf{GEN}}(x_t)$

1.4.3 Structured SVM

Structured SVM (SSVM) 32 is an extension of SVM for structured prediction problems. It is similar to MIRA except that SSVM is a batch learning algorithm with the following problem setting:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C \sum_t \xi_t$$

$$s.t. \quad \forall t : \ell(\boldsymbol{\theta}, (x_t, y_t)) \leq \xi_t, \ \xi_t \geq 0$$

where $\ell(\boldsymbol{\theta}, (x_t, y_t))$ is also the structured hinge-loss as in the MIRA case. The problem can be equivalently formulated as

$$\boldsymbol{\theta} = \underset{\boldsymbol{\theta} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{C}{2} \|\boldsymbol{\theta}\|^2 + \frac{1}{T} \sum_{t=1}^{T} \ell(\boldsymbol{\theta}, (x_t, y_t))$$

where T is the number of training samples.

Due to the large constraints in the SSVM problem setting, it usually does not permit closed-form solutions. Various optimization methods can be applied to find numerical solutions for this problem. For example, subgradient methods, cutting plane methods and bundle methods are a few of them. Note that SSVM is also often called Max-Margin Markov Network (M3N) 36. The two algorithms are very similar to each other, but SSVM emphasizes the hyper-plane view whereas M3N emphasizes its connection with probabilistic graphical models.

1.4.4 MERT

Minimum Error Rate Training (MERT) 29 was proposed by MT researchers and has been used almost exclusively by the MT community, although the general principle also applies to other tasks. Unlike the algorithms introduced in the previous sections, MERT does not have strong theoretical support and suffers from several drawbacks. However in practice it usually works well and has become a standard component of all major MT systems.

The motivation of MERT is to directly optimize the task-specific performance measure (say the BLEU score of MT outputs), which contrasts with previous examples where convex surrogate loss functions (say hinge-loss) are used as objectives. Since the error surfaces of direct costs (as a function of parameters) are in general non-smooth and non-convex, MERT optimizes parameters via line search. During each iteration, the algorithm searches along each dimension $\theta_i, i = 1, \ldots, d$ of the parameter space in turn while keeping other parameters $\theta_j, j \neq i$ fixed. At each search point $\theta_i = \tilde{\theta}_i$, the model makes corresponding predictions $y_1^0, \ldots, y_N^0$ for all inputs $x_1, \ldots, x_N$ based on the parameters $[\theta_1, \ldots \tilde{\theta}_i, \ldots, \theta_d]^T$. The best search point θ_i^* is the one whose corresponding predictions suffer from the least cost. Once θ_i^* is found, its value is fixed and another line search starts for θ_{i+1}. This procedure continues until the algorithm converges.

Although MERT usually performs well in practice, it suffers from the following problems: 1) No margin constraint; 2) Since the objective (direct cost) is usually non-convex, the performance is sensitive to the choice of initial parameter values; 3) Not scalable, cannot handle cases where the parameter dimension is higher than ~30. Therefore, despite its practical efficacy and popularity in the MT community, it is not regarded as a well-defined machine learning algorithm.

1.5 Overview and Contributions of the Thesis

In Chapter 2, we briefly go through background materials in online learning and information geometry.

In Chapter 3, we extend the multiplicative Winnow algorithm to a structured version, and the additive MIRA algorithm to its multiplicative version, then demonstrate empirical comparisons with their additive counterparts. We also give interpretations to the relationship between EG and `prod`, two multiplicative algorithms, from information geometric perspective.

In Chapter 4, we propose a new general framework named Generalized Multiplicative Update (GMU) that is multiplicative in nature and easily derives many multiplicative algorithms. We then propose a subclass of GMU, named the q-Exponentiated Gradient (qEG) method, that elucidates the relationship among several algorithms. Further analysis of these algorithms are given from Riemannian geometric perspective. We also extend OMD and GMU to accelerated versions by adding momentum terms.

In Chapter 5, we propose a novel approach for CRF training using natural gradient descent (NGD). The proposed framework is very flexible, allowing us to choose proper convex functions that lead to better training performance.

In Chapter 6, we propose an online learning algorithm by shifting from vector space to tensor space, which dramatically reduces the number of parameters to be estimated. The resulting model is highly regularized and is particularly suitable for training under low-resource environment.

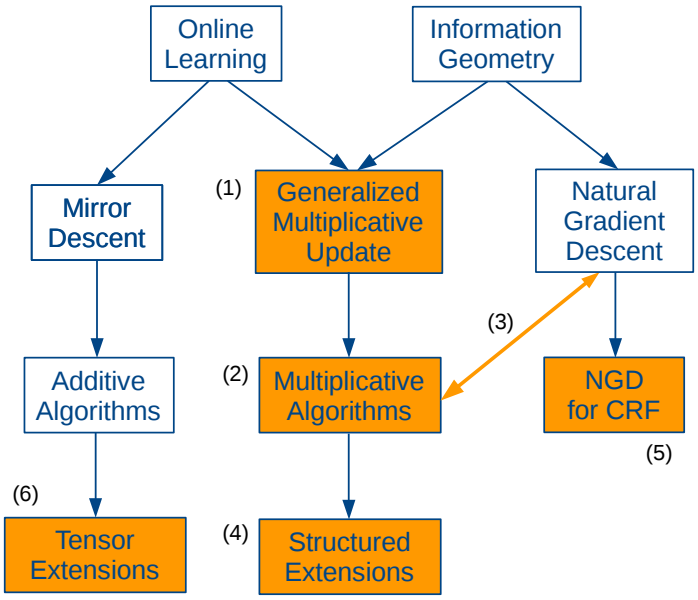


Figure 1.2: Summary of contributions: 1) Propose a General Multiplicative Update (GMU) framework (Chapter 4; 2) Develop new multiplicative learning algorithms (q-Exponentiated Gradient, Chapter 4); 3) Geometric interpretations to the relationship between EG and `prod` algorithms (Chapter 3); 4) Structured extensions to the Winnow and MIRA algorithms (Chapter 3); 5) Training CRF using NGD (Chapter 5); 6) Online learning in tensor space (Chapter 6).

The thesis is summarized in Chapter 7. Related concepts of convex analysis, Riemannian geometry, and divergences are given in Appendix A B C.

The contributions of the thesis is summarized in Figure 1.2 (yellow boxes).

Chapter 2

Background Review

In this chapter, we review some of the the technical backgrounds essential to the development of techniques proposed in later chapters.

2.1 Online Learning

The Perceptron, passive-aggressive and AdaGrad algorithms that we introduced in Chapter 1 are all efficient learning algorithms widely applied in NLP/ASR tasks. In fact, they are all special cases of a more general class of learing algorithms, named online learning (or online convex optimization). In this section, we give an overview of this fascinating class of learning algorithms.

Although the concept of online learning has been around for a long time, it was not until the last decade or so that rapid developments have taken place in this area, both in

theoretical foundations and practical applications. The overview presented in this section is by no means comprehensive. Interested readers are referred to monographs on this topic, for example, 59–66, for a more detailed introduction to this topic.

2.1.1 Problem Setting

Online learning can be treated as a repeated game between a learner and nature. At each round t of the game, the learner observes an instance $\mathbf{x}_t$ (say a feature vector associated with an object) and makes a prediction y_t' about the instance (say the class label of this object) based on this observation and his knowledge learned so far (say a weight vector $\boldsymbol{\theta}_t$ of the features). After the prediction is made, nature reveals the answer y_t and the learner incurs a loss $\ell_t = \ell_{\boldsymbol{\theta}_t}(\mathbf{x}_t, y_t)$ [1] if the prediction is incorrect. The learner then updates its knowledge based on the loss information, with the hope of making better predictions in future rounds of the game. The overall goal of the learner is to make the cumulative loss after T rounds of the game as small as possible. The online learning scheme is illustrated in Figure. 2.1.

In the setting of online learning, it is crucial to notice that no assumption about the distribution of the data is required. This is in sharp contrast with the classic statistical learning theory 67, where the world is assumed to be static: the training and test samples are assumed to be generated from the same distribution which does not change over time. Therefore in statistical learning theory, a model is learned via minimizing an empirical

[1] Note that the loss functional form may change over time, and the actual loss is also controlled by the parameter $\boldsymbol{\theta}_t$. A typical example is the squared loss $\ell_t = (\boldsymbol{\theta}_t^T \mathbf{x}_t - y_t)^2$ for regression.

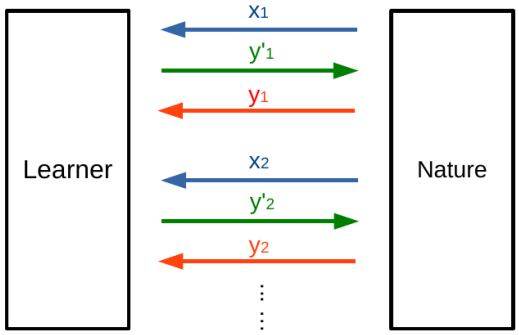


Figure 2.1: Online learning can be treated as a game between the learner and nature.

estimation of the expected loss, and the goal of the learner is to make sure that with high probability, the model will also make low prediction error on the test data. Online learning, on the other hand, discards the assumption of stationary distribution and aims to deal with the constantly changing world. That is to say, the samples $(\mathbf{x}_{t+1}, y_{t+1})$ observed at round $t+1$ can be generated from a totally different distribution than that of round t. In fact, the nature can even be adversarial: whatever that has been learned at round t may no longer be useful for round $t+1$ Since there is no stationary distribution generating the data, we no longer want to minimize the expectation of the error. In this sense, statistical learning and online learning are like two sides of a coin, making complementary assumptions about the data.

At first sight, online learning seems to be a hopeless problem since no matter what we do, there is always a way for nature to beat us. Therefore, a reasonable way of defining the goal of online learning is that the learner performs not too worse than the optimal predictor in hindsight. Note that the optimal predictor itself can be a bad predictor in terms of the absolute prediction accuracy due to the adversary of nature, however his performance is the

best we could expect and hence should be treated as our target. Given this problem setting, the performance of a learner is measured by *regret* in online learning, which is borrowed from game theory and defined as follows:

$$R_T = \sum_{t=1}^{T} \ell_t(\boldsymbol{\theta}_t) - \sum_{t=1}^{T} \ell_t(\boldsymbol{\theta}^*)$$

where $\boldsymbol{\theta}^*$ denotes the parameters of the optimal predictor (target vector) in hindsight, and $\ell_t(\boldsymbol{\theta})$ is the loss suffered by a predictor with parameters $\boldsymbol{\theta}_t$ at round t (note that the function ℓ_t may change with t). It can be seen that the regret measures how "sorry" the learner is, in retrospect, for not having followed the optimal predictor in hindsight. A properly designed online learning algorithm should never let the regret grow at $O(T)$, otherwise the regret would grow as fast as the number of rounds and learning would become meaningless. All online learning algorithms to be introduced in later sections have $O(\sqrt{T})$ or even $O(\log T)$ regret bounds, and this is what makes online learning interesting: no matter how the environment changes, eventually the learner is always able to catch up with the best predictor in hindsight.

Based on different levels of the availability of feedback information, online learning can be further divided into *full feedback* and *bandit feedback* settings. In the full feedback setting, all loss information is available to the learner (say the gradients of each dimension of the loss function), whereas in the bandit feedback setting, only the least information is provided (the learner only receives a single loss value after each round). On

the other hand, based on how one views the role of nature in the learning paradigm, the type of adversary in online learning can be classified as *oblivious* and *non-oblivious*. An oblivious adversary generates the loss ℓ_t only based on the learner's current prediction y_t, whereas a non-oblivious adversary generates the loss based on the learner's prediction history $y_1, y_2, \ldots, y_t$ (hence he is more adaptive and intelligent). More refined settings of online learning can be found, for example, in 68. In this thesis, we only consider the oblivious, full feedback setting. Introduction to the bandit setting algorithms can be found in, for example, 69.

2.1.2 Online Learning vs. Convex Optimization

As will be seen in Section 2.1.4, online learning (or *online* convex optimization) algorithms appear to be very similar to convex optimization algorithms, which makes the two concepts often confusable. Actually, many online learning algorithms did originate from convex optimization algorithms. For example, online gradient descent (OGD) proposed in 70, which in fact formally introduced the concept of online learning, has the same algorithmic form as the basic gradient descent as can be seen in Eq. 2.1, and the online mirror descent (OMD) framework (Eq. 2.2) is based on the mirror descent (MD) procedure proposed by Nemirovski 71 as early as in the 1980's.

However, despite the resemblance between online learning and convex optimization, they differ in the working environments and analysis frameworks. As introduced in Section 2.1.1, online learning allows the environment to be adversarial and the objective func-

tion may change over time. Therefore regret, which takes the entire prediction history or cumulative loss into consideration, is used as a performance measure. By contrast, general convex optimization procedure works on fixed objective functions, and in the case of stochastic convex optimization, a stationary distribution is assumed to have generated training samples, which makes the minimization of the expected loss meaningful. Accordingly, convergence rate is used as performance measure in convex optimization, in which case we only care about the end-point precision and how fast we may get there. The difference between regret and convergence rate as performance measures thus leads to different algorithmic analysis methodologies.

It can also be seen that convex optimization can be treated as a special case of online learning where the objective function is fixed. In this case there is a simple way to convert the regret bound to the convergence rate:

$$`(\bar{\boldsymbol{\theta}}_T) - `(\boldsymbol{\theta}^*) \leq \frac{1}{T}\sum_{t=1}^{T} `(\boldsymbol{\theta}_t) - `(\boldsymbol{\theta}^*) \leq \frac{R_T}{T}$$

where $`$ is the fixed objective, $\bar{\boldsymbol{\theta}}_T = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{\theta}_t$ serves as the solution after T^{th} iteration.

2.1.3 Online Learning vs. Batch Learning

The meaning of the word "online" can be interpreted in two ways. The first, as introduced in Section 2.1.1, refers to the distribution-free assumption which distinguishes online learning from statistical learning. The other interpretation is more concerned with

the learning protocol of the algorithms, which contrasts online learning with batch learning.

In batch learning, we need to first collect all the training data before starting the learning procedure, whereas in online learning each training sample is processed on-the-fly. The advantage of the online learning style is obvious: it does not require keeping a large amount of training data in memory, its update operation is usually light-weight and efficient, and the algorithm can be easily deployed in a changing environment where new training samples are always expected to come. All these features make online learning very fast and scalable. However, online learning may also take longer time to converge than batch learning, since individual sample points can be noisy and mislead the training procedure. However in the case of stochastic optimization, Léon Bottou 72 points out that although stochastic gradient descent (a special case of online learning) is not a good optimization algorithm, lower empirical risk can be achieved by processing more data within a fixed amount of time. Therefore, it is still a good learning algorithm when training time is the bottleneck. Nowadays as data sets are growing even larger and larger, scalability and efficiency of learning algorithms is a crucial measure of performance. Therefore, in practice we are more interested in developing and using online learning algorithms.

2.1.4 Basic Online Learning Algorithms

In this section, we briefly review some of the most important and fundamental online learning algorithms. Many more advanced online learning algorithms have been proposed in recent years (Section 2.1.5), however they can in general be treated as extensions to these

basic algorithms.

2.1.4.1 First-Order Methods

- *Online Gradient Descent (OGD) and related algorithms*

OGD 70 is a simple first-order method which has the following form of update:

$$\begin{aligned} \tilde{\boldsymbol{\theta}}_{t+1} &= \boldsymbol{\theta}_t - \lambda_t \partial \ell_t \\ \boldsymbol{\theta}_{t+1} &= \Pi_\Omega(\tilde{\boldsymbol{\theta}}_{t+1}) \end{aligned} \tag{2.1}$$

where $\partial \ell_t$, $\partial \ell_t(\boldsymbol{\theta}_t)$ is the subgradient of ℓ_t evaluated[2] at $\boldsymbol{\theta}_t$ and we use this notation throughout this thesis unless otherwise specified. Ω is a convex feasible set, and Π_Ω stands for projection onto the set Ω.

For this algorithm, we have the following regret analysis. Let G be the Lipschitz constant of ℓ_t, and D be the data diameter, namely $\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \leq D$. Setting $\lambda_t = \frac{G}{D\sqrt{t}}$, and the regret of OGD is given by:

$$R_T = \sum_{t=1}^{T} \ell_t(\boldsymbol{\theta}_t) - \min_{\boldsymbol{\theta}^* \in \Omega} \sum_{t=1}^{T} \ell_t(\boldsymbol{\theta}^*) \leq 3GD\sqrt{T}$$

When $\ell_t(\boldsymbol{\theta})$ is in addition α-strongly convex, by setting $\lambda_t = \frac{1}{\alpha t}$, the regret bound can be

[2]Strictly speaking, the subgradient $\partial f(\boldsymbol{w})$ of a convex function f at point $\boldsymbol{w}$ is the set of vectors $\boldsymbol{z}$ satisfying $f(\boldsymbol{u}) \geq f(\boldsymbol{w}) + \langle \boldsymbol{u} - \boldsymbol{w}, \boldsymbol{z} \rangle, \forall \boldsymbol{u}$ domain. Here we use the notation $\partial f(\boldsymbol{w})$ to represent a single element in this set. Also when f is differentiable, this becomes the gradient $\nabla f(\boldsymbol{w})$.

further improved from $O(\sqrt{T})$ to $O(\log T)$ [73]:

$$R_T \leq \frac{G}{2\alpha}(1 + \log T)$$

OGD is widely used in practice, because of its simplicity and low-regret guarantee. Actually it runs in linear time (if not considering the projection step) and it can be shown that the regret bound for first-order methods cannot be further improved [66].

Some popular convex optimization algorithms are tightly related to OGD. The stochastic gradient descent (SGD) [74, 75] can be treated as a special case of OGD, in the sense that the estimation of the objective (loss function) is stochastic (say the squared error loss function $\ell_t(\boldsymbol{\theta}) = (y_t - \boldsymbol{x}_t^T\boldsymbol{\theta})^2$ for input $(\boldsymbol{x}_t, y_t)$). The Perceptron algorithm described in Chapter1 is a special case of SGD where the loss function is the hinge-loss with zero margin. Another important and popular algorithm derived from SGD is Pegasos [76], which is used for efficient SVM optimization. The passive-aggressive/MIRA algorithm [23] optimizes the hinge loss online with margin constraint, which leads to updates similar to OGD but with adaptive learning rates λ_t.

2.1.4.2 General Learning Frameworks

Although a number of specific online learning algorithms have been proposed with different motivations, many of them can be subsumed in a few general learning frameworks, which we show below.

- *Follow-the-Regularized-Leader (FTRL)*

A very simple and intuitive way of learning online is to find the parameters after the t^{th} round by minimizing the cumulative losses of all previous rounds:

$$\boldsymbol{\theta}_{t+1} = \underset{\boldsymbol{\theta}\in\Omega}{\operatorname{argmin}} \sum_{i=1}^{t} \ell_i(\boldsymbol{\theta})$$

This learning scheme is named Follow-the-Leader (FTL), where the "leader" means the optimal predictor for all previous rounds 60, and it is just like batch-learning on-the-fly. This algorithm has a general regret bound:

$$R_T \leq \sum_{t=1}^{T} [\ell_t(\boldsymbol{\theta}_t) - \ell_t(\boldsymbol{\theta}_{t+1})]$$

However this algorithm is rarely used in practice as it does not have a regularization mechanism, which makes it unstable and easily disturbed by noisy training samples. Therefore, we would like to add a regularization term to the original FTL, which leads to the the Follow-the-Regularized-Leader (FTRL) algorithm:

$$\boldsymbol{\theta}_{t+1} = \underset{\boldsymbol{\theta}\in\Omega}{\operatorname{argmin}} \{\sum_{i=1}^{t-1} \ell_i(\boldsymbol{\theta}) + \lambda G(\boldsymbol{\theta})\}$$

where G is a convex function. The regret of FTRL is given as

$$R_T \leq \frac{1}{2\lambda}\|\boldsymbol{\theta}^*\|_2^2 + \frac{\lambda}{2}\sum_{t=1}^{T} \|\partial \ell_t\|_2^2$$

Further more, if $k\partial \grave{}_t k_2 \leq L$ and $k\boldsymbol{\theta}^* k_2 \leq B$, then a cleaner bound can be derived:

$$R_T \leq \frac{B^2}{2\lambda} + \frac{\lambda T L^2}{2}$$

Although the problem setting of FTRL is simple, it is not always the case that $\boldsymbol{\theta}_{t+1}$ can be efficiently computed. However, when the convex loss function $\grave{}_t$ is approximated via linearization, it can be easily shown that FTRL is equivalent to Online Mirror Descent (OMD) which we introduce next, whose update step is often simpler to compute.

- *Online Mirror Descent (OMD)*

Perhaps the most important and general framework of online learning algorithms, OMD is based on mirror-descent (MD) proposed by Nemirovski in the 1980's 71. Although MD was originally used for convex optimization, it was later introduced to online learning (see for example 77[3] 79) as OMD. OMD is treated as a gold standard for online learning algorithms, not only because many of the the proposed algorithms and their corresponding regret bounds can be recovered as special cases of OMD, but also it is in some sense optimal for online learning problems 80.

[3]In 77, the proposed general learning strategy is the same as OMD (even the regret bound is almost the same), but instead of using Bregman divergence as a proximity function, the algorithm proposed was directly induced by a "link function" (similar to the gradient of the convex fucntion in Bregman divergence). A brief overview of such early work of introducing general frameworks to online learning can be found in 78, for example.

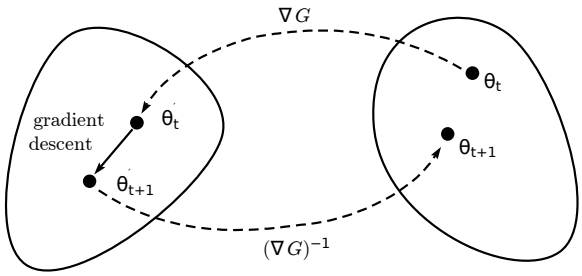


Figure 2.2: Conceptual plot of MD. Here $\boldsymbol{\theta}_t^0$, $\partial G(\boldsymbol{\theta}_t)$.

OMD has the following formulation:

$$\begin{aligned} \tilde{\boldsymbol{\theta}}_{t+1} &= \operatorname*{argmin}_{\boldsymbol{\theta}} \{ h\boldsymbol{\theta}, \partial_t i + \frac{1}{\lambda_t} D_G(\boldsymbol{\theta}, \boldsymbol{\theta}_t) \}, \\ \boldsymbol{\theta}_{t+1} &= \Pi_\Omega(\tilde{\boldsymbol{\theta}}_{t+1}), \end{aligned} \tag{2.2}$$

where D_G is the Bregman divergence (see Appendix C), and the update step (Eq.2.2) can be derived as

$$\nabla G(\boldsymbol{\theta}_{t+1}) = \nabla G(\boldsymbol{\theta}_t) - \lambda_t \partial \grave{}_t \tag{2.3}$$

A conceptual plot of OMD is given in Figure 2.2. It can be seen that OMD is equivalent to performing gradient descent in a dual space induced by the convex function G, followed by mirroring the updated dual parameters back to the primal space. OMD can be interpreted in different ways. First, it can be easily shown that OMD is equivalent to the FTRL using linearized loss function, that is using the first-order approximation $\tilde{\grave{}}_t(\boldsymbol{\theta}) = \grave{}_t(\boldsymbol{\theta}_t) + h\boldsymbol{\theta} - \boldsymbol{\theta}_t, \partial \grave{}_t(\boldsymbol{\theta}_t) i$ as the loss instead of $\grave{}_t$. It is common practice to linearize the loss in online learning algorithms since it makes the parameter update as well as the performance analysis

easier, and also gives an upper-bound of the true regret:

$$\sum_{t=1}^{T} (\ell_t(\boldsymbol{\theta}_t) - \ell_t(\boldsymbol{\theta}^*)) \leq \sum_{t=1}^{T} (\langle \boldsymbol{\theta}_t, \partial \ell_t(\boldsymbol{\theta}_t) \rangle - \langle \boldsymbol{\theta}^*, \partial \ell_t(\boldsymbol{\theta}^*) \rangle)$$

Another way to view OMD is that it is a special instance of proximal algorithms 81, whose details we omit here.

Assuming a fixed learning rate ($\lambda_t = \lambda$), the regret of OMD is given by

$$R_T \leq \frac{1}{\lambda}[G(\boldsymbol{\theta}^*) - G(\boldsymbol{\theta}_1)] + \frac{1}{\lambda}\sum_{t=1}^{T} D_{G^*}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\theta}_t)$$

where G^* is the convex conjugate of G. When the convex function G is α-strongly convex with respect to the norm $\|\cdot\|$, the regret bound can be further improved to

$$R_T \leq \frac{1}{\lambda}[G(\boldsymbol{\theta}^*) - G(\boldsymbol{\theta}_1)] + \frac{\alpha}{2\lambda}\sum_{t=1}^{T} \|\partial \ell_t\|_*^2$$

where $\|\cdot\|_*$ is the dual norm of $\|\cdot\|$.

Since we are free to choose G (or equivalently the Bregman divergence), OMD is very flexible and yields different specific online learning algorithms (although most of them were not proposed from this framework). Here we take a look at some of them:

1. Choose $G(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x}\|_2^2$, the $D_G(\boldsymbol{\theta}_{t+1}, \boldsymbol{\theta}_{t+1})$ is equivalent to the Euclidean distance

$\frac{1}{2}\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|_2^2$, and OMD recovers OGD (Eq.2.1), with the regret specialized to

$$R_T \leq \frac{1}{2\lambda}\|\boldsymbol{\theta}^*\|_2^2 + \frac{\lambda}{2}\sum_{t=1}^{T}\|\partial_t\|_2^2 \tag{2.4}$$

2. When $\boldsymbol{x}$ is constrained to be a $(d-1)$-simplex and we choose $G(\boldsymbol{x}) = \sum_{i=1}^{d} x_i \log x_i$ (namely the negative entropy), then $D_G(\boldsymbol{\theta}_{t+1}||\boldsymbol{\theta}_t)$ becomes the KL-divergence $\sum_{i=1}^{d}\theta_{t+1,i}\log\frac{\theta_{t+1,i}}{\theta_{t,i}}$, and the corresponding update strategy is given by

$$\theta_{t+1,i} = \frac{1}{Z_t}\theta_{t,i}\exp(-\lambda\partial_{t,i}), \ \forall i = 1, \ldots, d \tag{2.5}$$

where $Z_t = \sum_{i}^{d}\theta_{t,i}\exp(-\lambda\partial_{t,i})$ is the normalization term. This algorithm is named as exponentiated gradient descent (EG) 82. Note that unlike the OGD update which is *additive*, the update of EG is *multiplicative*. The corresponding regret bound is given by

$$R_T \leq \frac{\log(d)}{\lambda} + \frac{\lambda}{2}\sum_{t=1}^{T}\|\partial_t\|_\infty^2 \tag{2.6}$$

where $\|\partial_t\|_\infty \leq L$.

Comparing Eq. 2.4 with Eq. 4.20, we see that they have similar forms but use different norms in the bounds, and it is natural to ask which algorithm is more favorable in practice. The answer indeed depends on the nature of the problem: In cases where $\log(d) \approx \|\boldsymbol{\theta}^*\|_2^2$ but ∂_t is very dense, $\|\partial_t\|_\infty^2 \quad \|\partial_t\|_2^2$ and therefore the EG algorithm has a tighter bound. Otherwise, the OGD algorithm is a better choice. Such performance contrast caused by

different choices of norms also appears in other occasions of machine learning, for example SVM vs. Boosting 83, Truncated Gradient vs. SMIDAS 84, 85.

Note that when EG is applied for binary classification tasks, the corresponding algorithm is called Winnow 82, 86, 87 with the following update upon making a classification error:

$$\theta_{t+1,i} = \frac{1}{Z_t}\theta_{t,i}\exp(\lambda y_t x_{t,i}),\ \forall i = 1, \ldots, d \tag{2.7}$$

where $y_t \in \{-1, 1\}$ is the label for the t^{th} training sample.

3. Choose $G(\boldsymbol{x}) = \frac{1}{2(q-1)}\|\boldsymbol{x}\|_q^2$ where $q \geq 1$ and let $p = \frac{q}{q-1}$, the OMD update becomes

$$\begin{aligned}\tilde{\theta}_{t,i} &= \lambda\ \frac{sign(\theta_{t,i})|\theta_{t,i}|^{p-1}}{\|\boldsymbol{\theta}_t\|_p^{p-2}}\\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \partial\ell_t(\tilde{\boldsymbol{\theta}}_t)\end{aligned}$$

This recovers the p-norm gradient descent (similar to the p-norm Perceptron 88). The correpsonding regret becomes

$$R_T \leq \frac{1}{2\lambda(q-1)}\|\boldsymbol{w}\|_q^2 + \lambda\sum_{t=1}^{T}\|\partial\ell_t\|_q^2$$

When $q = 2$, the above algorithm is equivalent to the OGD, and when q is close to 1, the EG algorithm is recovered. Therefore, the p-norm update is a quasi-additive algorithm 89 interpolating between OGD and EG.

4. As a last example of OMD, we describe the Adaptive Subgradient Methods (AdaGrad) 28, which was proposed more recently and quickly gained popularity. We have seen that the selection of the convex function G is critical in the OMD framework: different choices of G lead to different algorithms and regret bounds. But what is the optimal choice of G that minimizes the regret? Not surprisingly, the optimal G should not be static, but rather be adaptive to the objective function and the convex feasible set.

This is exactly what AdaGrad 28 does: the convex function G_t is time-varying and updated online, giving a sequence of regularizers which adapts to the loss functions. It can be shown that AdaGrad reaches the minimum of the OMD regret bound (up to a constant). The function G_t is given by

$$G_t(\boldsymbol{\theta}) = h\boldsymbol{\theta}, H_t\boldsymbol{\theta}i,$$

$$\text{where} \quad H_t = \delta I + \text{diag}(K_t)^{\frac{1}{2}},$$

$$\text{and} \quad K_t = \sum_{i=1}^{t} \partial \grave{}_i \partial \grave{}_i^T,$$

and the update step can be derived from the OMD update formula[4]. The regret of AdaGrad is given by

$$R_T \leq 2D \min_{H \in H} \sqrt{\sum_t k\partial \grave{}_t k_H^{*2}}$$

[4]The original AdaGrad paper 28 proposed two styles of update: OMD and RDA (or primal-dual subgradient). Here we only present the OMD-style update for simplicity. Also, the full matrix of K_t instead of its diagonal can be used in the definition of H_t, with the sacrifice of computational efficiency.

where $D = \max_{\boldsymbol{u}\in\Omega} \max_{H\in\mathcal{H}} \|\boldsymbol{u}-\boldsymbol{\theta}_1\|_H$, $\mathcal{H} = \{X \in \mathbb{R}^{d\times d},\ tr(X) \le 1,\ X \succ 0\}$, $\|\cdot\|_H = \sqrt{\langle \cdot, H\cdot\rangle}$ is the Mahalanobis norm induced by the matrix H, and the learning rate λ is set to $\frac{1}{D}$.

- *Dual Averaging (DA)*

Another general framework for online learning is DA, proposed by Nesterov 90. DA is similar to OMD in the sense that it also considers primal and dual parameter spaces, however instead of using one control sequence (the step size λ_t, DA defines two control sequences, and the update formula is given by

$$\tilde{\boldsymbol{\theta}}_{t+1} = \tilde{\boldsymbol{\theta}}_t - \lambda_t \grave{}_t,$$

$$\boldsymbol{\theta}_{t+1} = \pi_{\beta_{t+1}}(-\tilde{\boldsymbol{\theta}}_{t+1}),$$

where $\pi_\beta(\boldsymbol{\theta}) \triangleq \operatorname{argmin}_{\boldsymbol{w}\in\Omega}\{-\langle \boldsymbol{\theta}, \boldsymbol{w}\rangle + \beta G(\boldsymbol{w})\}$ and $G(\boldsymbol{w})$ a convex function.

It can be seen that DA defines two control sequences λ_t, β_t, and the advantage of doing so is that unlike many other iterative procedures, the learning rate λ_t defined for the dual space (which accumulates the loss gradient information) is not required to shrink over time to converge or reach superior regret bound. The scaling between the primal and dual spaces can be taken care of by the β_t sequence. A simple choice of λ_t and β_t is $\lambda_t = 1$ and $\beta_t = \gamma\hat{\beta}_t$, where $\gamma > 0$ is a constant and $\hat{\beta}_t = \hat{\beta}_{t-1} + \frac{1}{\hat{\beta}_{t-1}}$, $\hat{\beta}_0 = \hat{\beta}_1 = 1$ Assuming $G(\boldsymbol{\theta})$ is α-strongly convex, the corresponding regret is given by

$$R_T \le \hat{\beta}_{t+1} \quad \gamma D + \frac{L^2}{2\alpha\gamma}$$

where $D \geq G(\boldsymbol{\theta}), \boldsymbol{\theta} \in \Omega$ and $\|\partial\|_* \leq L$.

Some extensions of the DA framework exists, for example 91, 92.

2.1.4.3 Second-Order Methods

All the methods we described above are instances of first-order methods (except AdaGrad, in which the second-order information is used). Not surprisingly, there also exist several second-order online learning algorithms, and we briefly describe one of them here, omitting all others.

- *Online Newton Step (ONS)*

The ONS proposed by 73 is basically a quasi-Newton algorithm adapted to online learning. The update step of ONS is given by

$$\boldsymbol{\theta}_{t+1} = \Pi_{\Omega}^{A_{t-1}} \left(\boldsymbol{\theta}_t - \frac{1}{\gamma} A_{t-1}^{-1} \partial_t \right) ,$$

where A_t is the smoothed covariance of gradients, namely $A_t = \sum_{i=1}^{t} \partial_i \partial_i^T + \boldsymbol{I}_n$, $= \frac{1}{\gamma^2 D^2}$, and $\Pi_{\Omega}^{A_{t-1}}$ is the projection using norm induced by A_{t-1}:

$$\Pi_{\Omega}^{A_{t-1}}(\boldsymbol{y}) = \operatorname*{argmin}_{\boldsymbol{y} \in \Omega} (\boldsymbol{y} - \boldsymbol{x})^T A_{t-1} (\boldsymbol{y} - \boldsymbol{x})$$

It can be seen that the Hessian of the objective is approximated by the covariance of its gradients, making the computation efficient.

The ONS algorithm performs well in the case of α-exp-concave loss functions. A

convex function $f(\boldsymbol{x})$ is said to be α-exp-concave if the function $g(\boldsymbol{x}) = \exp\{-\alpha f(\boldsymbol{x})\}$ is concave. Exp-concavity implies that the function is strongly convex in the direction of its gradient. For an α-exp-concave function, the regret of ONS is given by

$$R_T \leq 5\left(\frac{1}{\alpha} + GD\right) n \log T$$

2.1.5 More Advanced Algorithms

In Section 2.1.4 we have introduced some of the most fundamental online learning algorithms and frameworks. However, there are many more interesting algorithms that address specific issues with the basic algorithms which we did not cover here. In this section we briefly describe some of the important directions in which they extend the basic algorithms. Detailed algorithms can be found in Section2.1.6.

- *Composite Objective Functions*

The basic algorithms we presented in Section 2.1.4 treat the loss function as a blackbox, and all we need to have is an oracle that returns it gradient $\partial \ell_t$. However, in practice it is usually the case that the objective function has a structure, say it is a composition of a cost function and a regularization term: $\ell_t = f_t(\boldsymbol{\theta}) + k\boldsymbol{\theta}k_p$ where p is often 2 or 1. Although any general algorithm can be applied to optimize these objectives, exploring explicitly the structure of the objective may help to improve the performance. This is especially important in the case of ℓ_1 regularization, where a general procedure (say SGD) rarely finds a truly sparse solution. Therefore, extensions of these algorithms are proposed to find

better solutions specifically for composite objectives.

- *Adaptive/Time-varying Regularization*

Basic online learning adopts a fixed regularizer (convex proximity function) as can be seen in the case of FTRL, OMD and DA. However this limits the power of these algorithms as a fixed regularizer does not adapt to the actual running environment and may lead to suboptimal solutions. Therefore, it is sometimes desirable that we let the regularizer be time-varying and adaptive to the environment (see 59, Section 11.6). A typical example is the AdaGrad algorithm which we introduced in Section 2.1.4. Other time-varying algorithms have also been proposed recently.

- *Implicit Update*

Although linearizing the loss function is a convenient way of making the computation and analysis easier, as we see in Section 2.1.4, after all it is an approximation which may lead to suboptimal solutions. This problem becomes more severe when the loss function is strongly convex or the gap between $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t+1}$ is wide. To address this problem, algorithms that avoids linearizing the loss functions are proposed. In such cases, it is not always easy to find closed-form solutions for the updates, and the analysis becomes more difficult. However, there are some cases where we may get around this issue and give analysis to the performance.

- *Scale-free Algorithms*

From the regrets bounds given in Section 2.1.4 we see that they are usually controlled (in part) by the norm of the loss (or gradient) vectors. However, in practice it is often

impossible to know the upper-bounds of loss vector norm beforehand. Therefore, it is desirable to develop algorithms that are insensitive to the scale of loss vector norms, in other words the sequence of decisions made by the algorithms should not change if the vectors are scaled by a positive constant. Such algorithms are named scale-free algorithms.

- *Unifying Frameworks*

Although FTRL, OMD and DA are independently proposed and they provide nice and general frameworks for developing and analyzing online learning algorithms, it turns out that there are some interesting connections between them, and researchers have proposed even more general frameworks that elucidate their relationships and unify all of them.

2.1.6 Summary

We summarize the major online learning algorithms (basic and more advanced) in Table 2.1.

Table 2.1: A Summary of Online Learning Algorithms.

Type	Algorithms
Basic algorithms (additive)	OGD70, SGD75, Passive-Agressive23 Perceptron16, p-norm Perceptron (quasi-additive)88
Basic algorithms (multiplicative)	Winnow86, Weighted majority93 EG82, Hedge94, prod95
General framework	OMD71, FTRL64, DA90
Unifying framework	FTRL-proximal96, Ito's framework97
Composite objective	FOBOS98, COMID99, RDA91 FTRL-proximal96
Implicit update	Kulis' algorithm100, FTRL-proximal96
Adaptive algorithms	AOGD101, AdaGrad28, Adaptive MD102 DFEG103, AdaHedge104,
Scale-free	AdaFTRL/SOLOFTRL105
Second-order algorithms	ONS63, second-order Perceptron 106

Table 2.1 . . . continued

	CW/AROW/NAROW10726108, IELLIP109

In Chapter 4, we will take a deeper look into multiplicative online learning algorithms, and develop new techniques to interpret and develop this subclass of algorithms. Therefore, detailed introduction to important multiplicative algorithms like Hedge,`prod` will be deferred to later chapters. We also note that some important analysis techniques of online learning algorithms, for example in the case of strong convexity (for example 110), are ignored in this section.

2.2 Information Geometry

Information geometry is an elegant theory that marries Riemannian geometry with statistics. It studies the properties of the manifold of probability density functions (statistical manifold) using tools from differential geometry. The history of information geometry dates back to Rao's 1945 paper 111 in which the idea that statistical models can be considered as manifolds with the Fisher information matrix being the Riemannian metric was proposed. In 1975, Efron 112 introduced the notion of "statistical curvature" for 1-parameter models. In the same year, Dawid 113 invoked the flat connection in interpreting statistical models. Later, Eguchi 114 demonstrated that a well-defined divergence function induces a Riemannian metric and a pair of dual connections. Amari 115,116 systematically studied α-connections and manifolds with dually-flat structure which advanced the devel-

opment of this theory. In this section, we give a brief overview of information geometry. More detailed introductions to this topic can be found, for example, in 116–121.[5]

Since information geometry uses Riemannian geometry as the analysis tool, we give a brief introduction to the fundamentals of Riemannian geometry in Appendix B for readers not familiar with this topic.

2.2.1 General Ideas

The starting point of information geometry is to consider a family of probability distribution $S = \{p_{\boldsymbol{\theta}} = p(x; \boldsymbol{\theta}) | \boldsymbol{\theta} = [\theta_1, \ldots, \theta_n] \in \boldsymbol{\Theta}\}$ where $\boldsymbol{\theta}$ are the parameters and $\boldsymbol{\Theta}$ is a subset of $\mathbb{R}^n$, as a smooth C^∞ manifold (named statistical manifold). Given a distribution family S, a mapping $\phi : S \to \mathbb{R}^n$ defined by $\phi(p_{\boldsymbol{\theta}}) = \boldsymbol{\theta}$ allows us to consider $[\theta]$ as a coordinate system on the manifold. That is, each distribution in the family is abstracted as a point on the manifold. This is conceptually plotted in Figure 2.3. Based on this statistical

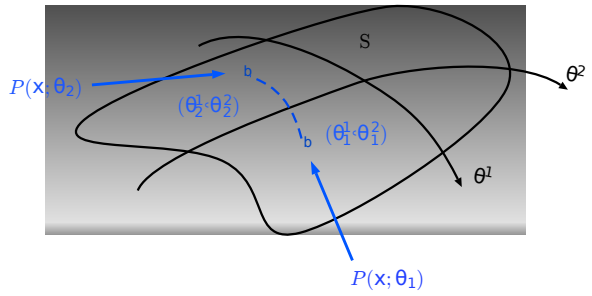


Figure 2.3: Conceptual plot of information geometry.

manifold, we are now able to borrow tools from differential geometry study the geomet-

[5] Although not included in the bibliography, many other papers written by Shunichi Amari and Frank Nielsen give nice introductions to information geometry from different perspectives, and they are highly recommended for reading.

ric properties of the manifold. The particular toolbox chosen by information geometry is the Riemannian geometry, which leads to some nice discoveries of geometric relationships between distributions.

A natural question to ask here is why Riemannian geometry? Why not other geometries, say the simple Euclidean geometry that we are so familiar with, or even advanced Finsler, symplectic geometries? In fact, we are free to make geometric assumptions about the statistical manifold, and the Riemannian geometry does not have to be the only tool of choice. However, whatever geometry we choose, the corresponding analysis has to make sense for statistical models. For example, the Riemannian geometric analysis makes better sense than the trivial Euclidean geometry, as some interesting and desirable properties of distribution families as we will see in the next a few sections cannot be derived from Euclidean geometry, and this makes it a suitable candidate for information geometry. Other geometries may also lead to interesting analysis, however we do not know much about their results yet.

To study the Riemannian geometric properties of distributions on the statistical manifold, we need to first introduce some structures to the manifold, most importantly the Riemannian metric tensor and affine connections. In information geometry, these structures are induced by properly defined divergence functions between distributions (in other words, a divergence function determines the geometry of a statistical manifold) [6]. While different divergences define different Riemannian geometries, some of them, notably the

[6] Although in Rao's early work 111, the use of the Riemannian metric (Fisher information) and connection (Levi-Civita) are directly proposed without the assistance from a divergence function.

f-divergence and Bregman divergence, are of particular interest as some nice and desirable geometric properties of distribution families can be derived. On the other hand, some distribution families, notably the exponential and mixture families, are privileged to have particularly simple geometric structures, making the analysis for these distributions easier.

In the following we give more details on the geometric properties of the statistical manifolds induced by divergence functions.

2.2.2 Divergence Functions and Geometry

A divergence defined on a statistical manifold S is a function $D(\cdot, \cdot) : S \times S \to \mathrm{R}$ satisfying

1. $D(x, y) \geq 0, \forall x, y \in S$
2. $D(x, y) = 0$ if and only if $x = y$.
3. The Hessian $G^{(D)}$ with components $g_{ij}^{(D)}$, $\partial_{x^i} \partial_{x^j} D(x, y)|_{x=y}$ is positive definite.

where the i^{th} parameter of x is denoted by x^i. Divergence functions serve as pseudo-distance measure between two points, but are not required to satisfy the axioms of distance (symmetry and triangle inequality). A typical example is the widely used Kullback-Leibler (KL)-divergence.

Divergence functions are tightly related to the geometry of manifolds. Eguchi 114 first demonstrated that Riemannian metrics and connections can be derived from divergences:

$$g_{ij}(x) = -\partial_{x^i}\partial_{y^j} D(x, y)\big|_{x=y} \tag{2.8}$$

$$\Gamma_{ij,k}(x) = -\partial_{x^i}\partial_{x^j}\partial_{y^k} D(x, y)\big|_{x=y} \tag{2.9}$$

$$\Gamma^*_{ij,k}(x) = -\partial_{y^i}\partial_{y^j}\partial_{x^k} D(x, y)\big|_{x=y} \tag{2.10}$$

where $\Gamma_{ij,k}$ and $\Gamma^*_{ij,k}$ determine *dual connections* ∇, ∇^*[7], which is defined by the following condition:

$$\partial_k g(\partial_i, \partial_j) = g(\nabla_{\partial_k}\partial_i, \partial_j) + g(\partial_i, \nabla^*_{\partial_k}\partial_j)$$

Therefore, a divergence function induces a Riemannian manifold S with dualistic structure: $\{S, g, \Gamma_{ij,k}, \Gamma^*_{ij,k}\}$. Indeed, a divergence can be regarded as a local distance measure on the manifold, which can be seen from its second-order approximation: $D(x, y) = \frac{1}{2} g_{ij}(y)\Delta\theta^i\Delta\theta^j + O(\|\Delta\boldsymbol{\theta}\|^2)$ where $\Delta\theta^i = x^i - y^i$. When y lies in an infinitesimal neighborhood of x, the residue $O(\|\Delta\boldsymbol{\theta}\|^2)$ can be ignored and $D(x, y)$ is equivalent to the squared distance between x, y on the manifold.

Although many divergence functions have been proposed in the literature, most of them are special cases of two general classes of divergences, namely the f-divergence and Bregman divergence. They both induce interesting geometric properties, which we describe next. More details about the two divergence classes can be found in Appendix C.

[7] Not to be confused with the gradient operator ∇.

2.2.2.1 f-Divergence and the Invariant Structure

The f-divergence, first proposed by Imre Csiszár 122, is defined as follows:

$$D_f(p, q) = \int p(x) f\left(\frac{q(x)}{p(x)}\right) dx \tag{2.11}$$

where $f(\cdot)$ is a strictly convex function satisfying $f(1) = 0$ (for discrete distributions, the integral is replaced with summation).

For two members p, q belonging to the same distribution family parameterized by $\boldsymbol{\theta}$, we may compute the Riemannian metric and connections induced by f-divergence according to Eq. 2.8– 2.10, which yields

$$g_{ij}(\boldsymbol{\theta}) = f''(1) E_{\boldsymbol{\theta}}[\partial_i \ell_p \partial_j \ell_p] = -f''(1) E_{\boldsymbol{\theta}}[\partial_i \partial_j \ell_p] \tag{2.12}$$

$$\Gamma_{ij,k}(\boldsymbol{\theta}) = \Gamma^{0}_{ij,k} - \frac{\alpha}{2} T_{ijk} \tag{2.13}$$

$$\Gamma^{*}_{ij,k}(\boldsymbol{\theta}) = \Gamma^{0}_{ij,k} + \frac{\alpha}{2} T_{ijk} \tag{2.14}$$

where ℓ_p , $\log p_{\boldsymbol{\theta}}(x)$, $g_{ij}(\boldsymbol{\theta})$ coincides with the Fisher information matrix $I(\boldsymbol{\theta})$ (up to a constant $f''(1)$), whose entries $I_{ij}(\boldsymbol{\theta})$ are also $E_{\boldsymbol{\theta}}[\partial_i \ell_p \partial_j \ell_p]$. In Eq. 2.13 and 2.14:

$$\Gamma^{0}_{ij,k} = \frac{1}{2}(\partial_i g_{jk} + \partial_j g_{ik} - \partial_k g_{ij}) = E_{\boldsymbol{\theta}}\left[\partial_i \partial_j \ell_p + \frac{1}{2}\partial_i \ell_p \partial_j \ell_p\right] \partial_k \ell_p$$

is the Levi-Civita connection with respect to the Fisher information metric [8], $\alpha = 3 + 2f'''(1)$ and $T_{ijk} = E_{\boldsymbol{\theta}}[\partial_i \grave{}_p \partial_j \grave{}_p \partial_k \grave{}_p]$. The connections $\Gamma_{ij,k}(\boldsymbol{\theta})$ and $\Gamma^*_{ij,k}(\boldsymbol{\theta})$, denoted by $\Gamma^{\alpha}_{ij,k}$ and $\Gamma^{-\alpha}_{ij,k}$ respectively, are dually coupled connections named α-connections. It can also be seen that the Levi-Citiva connection is the average of the dual connections: $\Gamma^0_{ij,k} = \frac{1}{2}(\Gamma^{\alpha}_{ij,k} + \Gamma^{-\alpha}_{ij,k})$.

Therefore, we see that f-divergence always induces a *Fisher information metric* + α-*connection* manifold structure $\{S, \tilde{g}, \Gamma^{\alpha}_{ijk}, \Gamma^{-\alpha}_{ijk}\}$, and the choice of f only affects the scale of the metric and the value of α.[10] The $\{S, \tilde{g}, \Gamma^{\alpha}_{ijk}, \Gamma^{-\alpha}_{ijk}\}$ structure (hence the f-divergence) is important as it is the only metric and connection that is invariant with respect to an invertible mapping of the variables. More specifically, let $y = m(x)$ where $m(\cdot)$ is an invertible function, we get a new distribution $\hat{p}(y; \boldsymbol{\theta}) = \frac{1}{\left|\frac{\partial y}{\partial x}\right|} p(m^{-1}(y); \boldsymbol{\theta})$. Ideally we hope that the geometric structure of the statistical manifold before and after the mapping remains the same, since the geometry should reflect the intrinsic properties of the distributions. It can be shown that the Fisher information metric and the α-connection meet this requirement, and in fact they are the only structure that satisfies this invariance criterion, which is known as the Chentsov theorem 123. Therefore the geometry $\{S, \tilde{g}, \Gamma^{\alpha}_{ijk}, \Gamma^{-\alpha}_{ijk}\}$

[8] To see this, notice that $\partial_k g_{ij} = \int_{\mathcal{X}} \partial_k \partial_i \grave{}_p \partial_j \grave{}_p dx + \int_{\mathcal{X}} \partial_i \grave{}_p \partial_k \partial_j \grave{}_p dx + \int_{\mathcal{X}} \partial_i \grave{}_p \partial_j \grave{}_p \partial_k p dx$, and $\partial_k p = \partial_k \grave{}_p p$.

[9] The connection coefficients $\Gamma^0_{ij,k}$ here are called Christoffel symbols of the first kind. A symmetric definition is the Christoffel symbols of the second kind. See the Levi-Civita connection introduced in Appendix B.4.

[10] It can be seen that only when $\alpha = 0$ is the connection Levi-Civita, which means that in general (when $\alpha \neq 0$) a "straight" line on the manifold defined by parallel transport is not the geodesics between two points. However, the study of α-connection instead of Levi-Civita is indeed a breakthrough in information geometry, as the analysis of the Levi-Civita connection (Rao's approach) usually does not admit closed-form solutions, impeding further explorations. The study of α-connection (Amari's approach) on the other hand, makes the analysis easier and reveals many interesting geometric properties as we will see shortly.

induced by the f-divergence is often known as the *invariant structure* due to its uniqueness in satisfying this criterion. What is more, the f-divergence also satisfies the information monotonicity criterion, which we do not elaborate here but more details can be found in Appendix C. These are all desirable properties for statistical manifolds, making f-divergence a very special and interesting divergence for geometric analysis.

Among the members of f-divergence family, an important instance is the α-divergence, defined as[11]

$$D_{(\alpha)}(p, q) = \frac{4}{1-\alpha^2}\left(1 - \int p(x)^{\frac{1-\alpha}{2}} q(x)^{\frac{1+\alpha}{2}} dx\right) \tag{2.15}$$

where $\alpha \in \mathbb{R}$ is a free parameter. When $\alpha = -1$ or 1, $D_{(\alpha)}(p, q)$ converges to $KL(p, q)$ and $KL(q, p)$ respectively. The α-divergence can be recovered from the f-divergence by choosing the following f:

$$f(x) = \begin{cases} \frac{4}{1-\alpha^2}\left(1 - x^{\frac{1+\alpha}{2}}\right) & \alpha \neq \pm 1 \\ x \log x & \alpha = 1 \\ -\log x & \alpha = -1 \end{cases} \tag{2.16}$$

α-divergence is an important case as it can be easily verified that $\frac{3+\alpha}{2} f'''(1) = \alpha$, which coincides with the connection parameter α in the definition of Γ^{α}_{ijk} (and $\Gamma^{-\alpha}_{ijk}$). In other words, the $\{S, g, \Gamma^{\alpha}_{ijk}, \Gamma^{-\alpha}_{ijk}\}$ structure can be directly determined by this single divergence

[11] In history the α-divergence has been proposed independently by different researchers, it has other names and variations with similar functional forms. See Appendix C.1.1 for a brief summary.

function.

2.2.2.2 Bregman Divergence and the Dually Flat Structure

The analysis of the geometric properties induced by Bregman divergence relies heavily on convex analysis, and it is amazing that the two independently developed fields find an intersection here. As will be seen shortly, the reason why convex analysis is important is that the Bregman divergence induces dually flat manifolds equipped with dually affine coordinate systems. The two coordinate systems can be transformed to each other via convex conjugate functions, and the function pairs happen to preserve local metrics under different coordinate systems. For readers not familiar with convex analysis, a brief introduction is given in Appendix A.

The Bregman divergence, proposed by Lev M. Bregman 124, is defined as follows:

$$D_G(p, q) = G(p) - G(q) - \nabla G^T(q)(p - q) \tag{2.17}$$

where $G(\cdot)$ is a strictly convex function and $\nabla G(q)$ denotes the gradient of $G(q)$. Meanwhile, a dual version of D_G can be similarly defined as

$$D_H(p, q) = H(p) - H(q) - \nabla H^T(q)(p - q) \tag{2.18}$$

where $H(\cdot)$ is the convex conjugate of $G(\cdot)$[12]. The conjugate function pairs also admit two

[12]The Bregman divergence can be equivalently defined in its dual form using convex conjugate functions

dual coordinate systems. Let $\boldsymbol{\theta}_x$ be the coordinate of a point x on the manifold, its dual coordinate is defined via the Legendre transformation: $\boldsymbol{\eta}_x = \nabla G(\boldsymbol{\theta}_x)$. Similarly, we may define the dual of $\boldsymbol{\eta}_x$ as $\hat{\boldsymbol{\theta}}_x = \nabla H(\boldsymbol{\eta}_x)$ and the relation $\hat{\boldsymbol{\theta}}_x = \boldsymbol{\theta}_x$ holds. What is more, it is also easy to verify that $D_G(\boldsymbol{\theta}_p, \boldsymbol{\theta}_q) = D_H(\boldsymbol{\eta}_q, \boldsymbol{\eta}_p)$.

Now that we have a pair of dual coordinate systems $\boldsymbol{\theta}, \boldsymbol{\eta}$ and Bregman divergences $D_G(\boldsymbol{\theta}_p, \boldsymbol{\theta}_q)$, $D_H(\boldsymbol{\eta}_q, \boldsymbol{\eta}_p)$, it is natural to consider both of them when studying the geometry induced by the Bregman divergences. Following Eq. 2.8– 2.10, we get the following metrics and connections[13]:

$$g_{ij}^{(D_G)}(\boldsymbol{\theta}) = \frac{\partial^2 G(\boldsymbol{\theta})}{\partial\theta^i \partial\theta^j}, \quad g_{ij}^{(D_H)}(\boldsymbol{\eta}) = \frac{\partial^2 H(\boldsymbol{\eta})}{\partial\eta^i \partial\eta^j} \tag{2.19}$$

$$\Gamma_{ij,k}^{(D_G)}(\boldsymbol{\theta}) = 0, \quad \Gamma_{ij,k}^{(D_G)*}(\boldsymbol{\theta}) = \frac{\partial^3 G(\boldsymbol{\theta})}{\partial\theta^i \partial\theta^j \partial\theta^k} \tag{2.20}$$

$$\Gamma_{ij,k}^{(D_H)}(\boldsymbol{\eta}) = \frac{\partial^3 H(\boldsymbol{\eta})}{\partial\eta^i \partial\eta^j \partial\eta^k}, \quad \Gamma_{ij,k}^{(D_H)*}(\boldsymbol{\eta}) = 0 \tag{2.21}$$

What makes the induced geometry remarkable are the following facts:

1. It can be shown that $g_{ij}^{(D_G)}(\boldsymbol{\theta}) = 1/g_{ij}^{(D_H)}(\boldsymbol{\eta})$, which yields the following result: $ds^2 = \sum g_{ij}^{(D_G)}(\boldsymbol{\theta}) d\theta^i d\theta^j = \sum g_{ij}^{(D_H)}(\boldsymbol{\eta}) d\eta^i d\eta^j$, or equivalently $M^{(D_G)} = M^{(D_H)^{-1}}$ where $M^{(D_G)}$ and $M^{(D_H)}$ are the Riemannian metric matrices induced by G and H respectively (see Eq.A.3). That is to say, the Riemannian metric is preserved when shifting from one coordinate system to the other.

(see Fenchel-Young inequality in SectionA.2): $D_{G,H}(p, q) = G(p) + H(q) - \sum_i p^i q^i$ (assuming discrete p, q here). This is known as the canonical divergence (see for example Section 3.4 in 116).

[13] From the metric and connections given above it can be seen that the geometry is completely determined by a convex function G (or H). Therefore, inducing geometry from a Bregman divergence is no different than from a convex function (and its convex conjugate for a dual geometry).

2. It can be shown that for $\Gamma_{ij,k}^{(D_G)} = 0$, the corresponding curvature tensor vanishes, which further implies that the dual curvature with respect to $\Gamma_{ij,k}^{(D_G)*}$ also vanishes (see Theorem 3.3 in 116). Therefore the manifold defined by D_G is flat with respect to both dual connections, and such a structure $\{S, g, \Gamma_{ij,k}, \Gamma_{ij,k}^{*}\}$ is called *dually flat*. The same can be said about the dual manifold induced by D_H.

For a dually flat manifold, the dual coordinate systems $\boldsymbol{\theta}, \boldsymbol{\eta}$ are affine and orthogonal with respect to the Riemannian metric: $h\frac{\partial}{\partial\theta_i}, \frac{\partial}{\partial\eta^j}i = \delta_i^j$. What is more, the geodesics are linear in both coordinate systems, given by $\boldsymbol{\theta}(t) = t\boldsymbol{a} + \boldsymbol{b}$ and $\boldsymbol{\eta}(t) = t\boldsymbol{c} + \boldsymbol{d}$ respectively (where $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}$ are constant vectors). These properties simplifies the geometric structure and makes the analysis easier.

Thanks to the special structure of the dually flat manifold, some interesting geometric properties can be derived, most notably the generalized Pythagorean theorem and the projection theorem. We do not elaborate on this topic here and interested readers may find details in many information geometry literatures (for example Section 3.4 in 116). These properties have intuitive explanations and have been used to give geometric interpretations to many machine learning algorithms[14].

2.2.2.3 The Uniqueness of the KL-Divergence

As we have seen, both f-divergence and Bregman divergence induce their own geometries. Therefore, a divergence that lies in the intersection of both classes may inherit all the

[14]Most commonly, problems involving constrained optimization are interpreted as geodesic projection onto a submanifold.

interesting geometric properties given by both sides. KL-divergence is such an instance, which can be derived from f-divergence by setting $f(x) = x \log x$ and from Bregman divergence by letting G be the negative entropy. Therefore, the geometry of KL-divergence is both invariant (from f-divergence) and dually flat (from Bregman divergence). In fact, for probability measures the KL-divergence is the *unique* intersection of the f and Bregman divergences 125, making it a privillaged candidate having very well defined geometric properties.

It should also be noted that unlike the Bregman divergence, the manifold induced by the f-divergence is in general not flat since the α-connections (Eq.2.132.14) is in general not 0. It is only in the case of KL-divergence that the f-divergence induces flat geometry, borrowing properties from Bregman divergence.

2.2.3 Geometry of Exponential and Mixture Families

Having introduced the geometries induced by the f-divergence and Bregman divergence, we now take a look at their effects on two specific distribution families: the exponential and mixture families. More detailed introduction to the properties of exponential families in the context of information geometry can be found in 126.

As one of the most important distribution families, the exponential family parameter-

ized by $\boldsymbol{\theta} \in \mathbb{R}^n$ has the following form:

$$p(x; \boldsymbol{\theta}) = \exp\left[C(x) + \sum_{i=1}^{n} \theta^i \varphi_i(x) - A(\boldsymbol{\theta})\right]$$

where $A(\boldsymbol{\theta}) = \log \int \exp\left[C(x) + \sum_{i=1}^{n} \theta^i \varphi_i(x)\right] dx$ is the partition function, $C(x)$ is the carrier measure and $\varphi_i(x)$ the sufficient statistics. The partition function $A(\boldsymbol{\theta})$ is particularly important as it contains a lot of information about a distribution in the exponential family. Specifically,

$$\nabla A(\boldsymbol{\theta}) = E_{\boldsymbol{\theta}}[\boldsymbol{\Phi}] \tag{2.22}$$

$$\nabla^2 A(\boldsymbol{\theta}) = \Sigma_{\boldsymbol{\theta}}[\boldsymbol{\Phi}] = M(\boldsymbol{\theta})$$

where $\Sigma_{\boldsymbol{\theta}}[\boldsymbol{\Phi}]$is the covariance matrix of the sufficient statistics vector, $M(\boldsymbol{\theta})$ is the Fisher information matrix with each entry $M_{ij}(\boldsymbol{\theta}) = E_{\boldsymbol{\theta}}[\partial_i \ell_p \partial_j \ell_p]$.

Now let us see what geometry the f-divergence brings to the exponential family. The manifold induced by the f-divergence is in general not flat, however by setting $\alpha = 1$we have the following connection: $\Gamma^{(1)}_{ij,k} = E_{\boldsymbol{\theta}}[\partial_i \partial_j \ell_p \partial_k \ell_p]$. Note that for exponential families the term $\partial_i \partial_j \ell_p = -\partial_i \partial_j A(\boldsymbol{\theta})$ is a constant, and it can be easily verified that $E_{\boldsymbol{\theta}}[\partial_k \ell_p] = 0$ Therefore $\Gamma^{(1)}_{ij,k} = -\partial_i \partial_j A(\boldsymbol{\theta}) E_{\boldsymbol{\theta}}[\partial_k \ell_p] = 0$, thus the corresponding manifold is flat and $\boldsymbol{\theta}$ is an affine coordinate system! The corresponding connection $\nabla^{(1)}$ is thus called *e-connection* (exponential-connection). In this case, the corresponding divergence must be

the KL-divergence since it is the only member in the f-divergence class that induces flat geometry.

Since the KL-divergence is also a member of the Bregman divergence, the exponential family manifold also inherits the dually flat properties of the Bregman divergence:

1. *Dually affine coordinate systems*: The convex function G of the Bregman divergence that yields the KL-divergence for the exponential family happens to be $A(\boldsymbol{\theta})$. According to the Legendre transformation $\boldsymbol{\eta} = \nabla A(\boldsymbol{\theta})$ which defines a map between two coordinate systems, and together with Eq.2.22, we easily find the dual affine coordinate system for the manifold of exponential families: $\boldsymbol{\eta} = E_{\boldsymbol{\theta}}[\boldsymbol{\Phi}]$ On the other hand, to map from $\boldsymbol{\eta}$ back to $\boldsymbol{\theta}$, we need to perform Legendre transformation on the convex conjugate $A^*(\boldsymbol{\eta})$ of $A(\boldsymbol{\theta})$. It can be shown that $A^*(\boldsymbol{\eta}) = -H_{\boldsymbol{\eta}}(p)$ where $H_{\boldsymbol{\eta}}(p)$ is the entropy of the distribution p (note that $\boldsymbol{\eta}$ is an implicit function of $\boldsymbol{\theta}$ satisfying $\boldsymbol{\eta} = E_{\boldsymbol{\theta}}[\boldsymbol{\Phi}]$). Therefore, the inverse coordinate mapping is given by $\boldsymbol{\theta} = -\nabla H_{\boldsymbol{\eta}}(p)$. $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ are known as the natural and expectation parameters of exponential families respectively, and the dual coordinate systems have been applied to solving problems like variational inference 127.

2. *Linear geodesics*: The geodesics between two points on the exponential family dual manifolds have the following linear form:

$$L(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2) = \{\boldsymbol{\theta} = (1 - t)\boldsymbol{\theta}_1 + t\boldsymbol{\theta}_2 | t \in [0, 1]\}$$

$$L^*(\boldsymbol{\eta}_1, \boldsymbol{\eta}_2) = \{\boldsymbol{\eta} = (1 - t)\boldsymbol{\eta}_1 + t\boldsymbol{\eta}_2 | t \in [0, 1]\}$$

Since by setting $\alpha = 1$ the geometry is particularly interesting for exponential families, it is natural to ask what would happen when $\alpha = -1$. It turns out that in this case similar geometric properties can be derived for another distribution family, namely the mixture family, defined as follows:

$$p(x; \boldsymbol{\theta}) = C(x) + \sum_{i=1}^{n} \theta^i F_i(x)$$

where $C(x)$ and $F_i(x)$ are two sets of functions, and θ^i are the mixture parameters. A typical member of the mixture family is the mixture of $n + 1$ probability distributions. The analysis procedure and results are similar to that of the exponential families, and we omit the details here. For $\alpha = -1$ with mixture families, the corresponding connection $\nabla^{(-1)}$ is called *m-connection* (mixture-connection).

Finally, we take a look at the geometry of discrete distributions, which are distributions defined on finite sets $X = \{1, \ldots, n\}$. The set of all distributions $S_n = \{p(x)\}$ forms an $(n - 1)$-dimensional manifold ($(n - 1)$-simplex) with parameters $p_i = p(x = i) > 0$. Discrete distributions belong to both exponential and mixture families as follows:

As exponential family:

$$p(x; \boldsymbol{\theta}) = \exp\left(\sum_{i=1}^{n-1} \theta^i \delta_{x,i} - A(\boldsymbol{\theta}) \right)$$

where $\theta^i = \log p_i - \log p_n$, $A(\boldsymbol{\theta}) = \log \left(1 + \sum_{i=1}^{n-1} \exp(\theta^i)\right) = -\log p_n$. The convex

conjugate of $A(\boldsymbol{\theta})$ is just the negative entropy: $A^*(\boldsymbol{\eta}) = \sum_{i=1}^{n} p_i \log p_i$.

As mixture family:

$$p(x; \boldsymbol{\theta}) = \sum_{i=1}^{n-1} \theta^i \delta_{x,i} + \theta^n \delta_{x,n}$$

where $\theta^i = p_i$.

From the perspective of f-divergence, the Riemannian metric (Fisher information) of S_n is given by $g_{ij} = \frac{1}{p_i}\delta_{i,j}$, and the dual connections are $\Gamma^{\alpha}_{ijk} = \frac{1}{2p_i}(-1-\alpha)\delta_{i,j,k}$, $\Gamma^{-\alpha}_{ijk} = \frac{1}{2p_i}(-1+\alpha)\delta_{i,j,k}$. When $\alpha = 1$ or -1, S_n is flat and equipped with dually affine coordinate systems linked by the Legendre transformation. The primal coordinates are given by $\theta^i = \log p_i - \log p_n$, and the dual coordinates $\eta^i = E_{\boldsymbol{\theta}}[\delta_{x,i}] = p_i$.

2.2.4 Natural Gradient Descent

Natural gradient descent (NGD) 128 is derived from the analysis of information geometry, and is one of its most popular applications. The conventional gradient descent (GD) assumes the underlying parameter space to be Euclidean, nevertheless this is often not a proper assumption (say when the space is a statistical manifold). By contrast, NGD shifts GD to non-Euclidean parameter spaces and takes the Riemannian geometry of the space into consideration.

Let $M = \{\boldsymbol{\theta} \in \mathrm{R}^n\}$ be the parameter space on which an objective function $L(\boldsymbol{\theta})$ is defined. Assuming that M is a Riemannian manifold, where the squared length of a small

incremental vector $d\boldsymbol{\theta}$ connecting $\boldsymbol{\theta}$ and $\boldsymbol{\theta} + d\boldsymbol{\theta}$ is given by

$$\|d\boldsymbol{\theta}\|^2 = \sum_{i,j} g_{ij}(\boldsymbol{\theta}) d\theta^i d\theta^j$$

where g_{ij} is the Riemannian metric. To do gradient descent on this manifold, we need to find the steepest descent direction $d\boldsymbol{\theta}$ that minimizes $L(\boldsymbol{\theta} + d\boldsymbol{\theta})$ where $d\boldsymbol{\theta}$ has a small fixed length ε. Let $d\boldsymbol{\theta} = \varepsilon \boldsymbol{w}$ with the constraint $\|\boldsymbol{w}\| = 1$ and use the first-order approximation $L(\boldsymbol{\theta} + d\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}) + \varepsilon \nabla L(\boldsymbol{\theta})^T \boldsymbol{w}$. Using the Lagrangian method

$$\frac{\partial}{\partial w^i} \left[\nabla L(\boldsymbol{\theta})^T \boldsymbol{w} - \rho \boldsymbol{w}^T G(\boldsymbol{\theta}) \boldsymbol{w} \right] = 0$$

we get $d\boldsymbol{w} = \frac{1}{2\rho} G(\boldsymbol{\theta})^{-1} \nabla L(\boldsymbol{w})$ where $G(\boldsymbol{\theta})$ is the Riemannian metric at point $\boldsymbol{\theta}$, and ρ can be absorbed by the learning rate. Therefore the steepest descent direction (natural gradient) is given by $G(\boldsymbol{\theta})^{-1} \nabla L(\boldsymbol{w})$, and the NGD update has the form

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t G(\boldsymbol{\theta})^{-1} \nabla L(\boldsymbol{\theta}) \tag{2.23}$$

In the special case of Euclidean geometry, the Riemannian G is just the identity matrix and the GD algorithm can be recovered: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t \nabla L(\boldsymbol{\theta})$. When the objective functions are probabilistic models, the parameters are those that controls distribution families. In this case, the analysis of information geometry shows that Riemannian manifold with Fisher information as the metric is a good assumption about the geometry of the parameter

space, and the NGD becomes $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t M(\boldsymbol{\theta})^{-1} \nabla L(\boldsymbol{\theta})$. It is shown in 128 that NGD with the Fisher information metric is asymptotically efficient and often converges faster than GD. Despite the nice properties of NGD, it is in general difficult to implement due to the nontrivial computation of M^{-1}. Researchers have proposed techniques to simplify, approximate, or even bypass the computation of M^{-1}, for example 129–132, by taking advantages of specific problem structures.

2.2.5 Summary

Information geometry studies the geometric properties of the statistical manifold. In the early days of information geometry, the Fisher information metric and Levi-Civita connection were directly used to study the geometry (sometimes called the Rao-manifold), which makes further analysis difficult. This motivated researchers to shift the focus from studying Levi-Civita to the dual α-connections (sometimes called the Amari-Chentsov manifold), which significantly promoted the development of this subject. It turned out that the geometry of the statistical manifold is tightly related to divergence functions, as the Riemannian metrics and dual connections can be directly induced from divergences. As a consequence, two important general divergence classes, namely the f-divergence and Bregman divergence have been carefully studied. They each induce interesting geometric properties, summarized as follows:

- *f-divergence*

Fisher information metric + dual α-connection, invariant structure, in general not flat.

- *Bregman divergence*

Riemannian metric, dual connections defined by a convex function, dually flat structure, dually affine coordinate systems, linear geodesics, Pythagorean and projection theorems.

For special distributions like the exponential and mixture families, some interesting properties can be derived from the geometric properties of both divergences.

Although information geometry and online learning were developed by different group of researchers and the two subjects seem irrelevant at first sight, it indeed provides us with a different angle to look at online learning algorithms and inspires us to develop novel learning algorithms, as we will see in Chapter 4.

Chapter 3

Multiplicative Training Methods

3.1 Introduction

As we have seen in Section 2.1.4, there are in general two types of online learning update strategies: additive and multiplicative. The OGD (Eq. 2.1) is a typical additive algorithm, and the EG (Eq. 2.5) is a typical multiplicative algorithm. In this chapter, we give a more detailed introduction to this interesting subclass of online learning algorithms (Section 3.2), and give interpretations to the relationship between EG and `prod` algorithms from information geometric perspective 3.3. We then propose two extensions to existing methods, namely the Structured Winnow and Multiplicative-MIRA algorithms (Section 3.4) and compare their performance with their additive counterparts.

3.2 Multiplicative Online Learning Algorithms

In terms of the update styles, there are two types of multiplicative online learning algorithms proposed in literature so far: the EG-style algorithms (error information gets exponentiated) and the `prod`-style algorithms (error information adjusted by a linear function) 133, 134.[1] In this section, we take a look at some examples of the multiplicative learning algorithms: the Weighted Majority, Hedge and `prod`. The Weighted Majority and Hedge algorithm adopt the EG-style update, and the `prod` adopts the `prob`-style update. More comprehensive introductions of these algorithms, including their applications and extensions, can be found in 94, 133, 136–139.

- Weighted Majority (WM)

The WM algorithm was proposed by 93, and is a simple algorithm based on the strategy of "prediction with expert advice". Suppose we would like to perform a sequence of binary classification task, and there are d experts available to us (say d weak classifiers or feature functions), each of which will give his own prediction. We then make our final decision based on the "advices" from these experts. Just like in the online learning setting, the nature can be adversarial and the sequence of samples does not have to be drawn from a fixed distribution. Some experts do better than other experts in terms of prediction accuracy, but we do not know which one is the best (otherwise we just need to follow that expert all the time). However we would like to develop an algorithm whose predictions are at least not

[1] The `prod` algorithm is called "multiplicative weights algorithm" in 133, and is called "polynomial weights algorithm" in 135. In this thesis we adopt the name `prod` as proposed in 95.

too far from those given by that best expert in hindsight, and the WM is a simple algorithm that achieves this goal. The WM algorithm makes prediction according to the weighted votes from each expert, and updates the weights based on the cost suffered by each expert after the correct label is revealed. More precisely, let $\theta_{t,i} \geq 0$ be the weight assigned to the i^{th} expert at round t, $e_{t,i} \in \{0, 1\}$ be the prediction made by expert i at round t, $m_{t,i}$ be the cost suffered by expert i after the correct label is revealed, and $\beta \in [1/2, 1)$ a constant, then the WM algorithm proceeds as follows[2]:

Initialization:

$$\theta_{0,i} = 1, \quad \forall i = 1, \ldots, d$$

At round t:

$$v_{t,0} = \sum_{i:e_{t,i}=0} \theta_{t,i}, \; v_{t,1} = \sum_{i:e_{t,i}=1} \theta_{t,i}$$

If $v_{t,0} > v_{t,1}$, predict class label 0; Otherwise predict 1.

Update weights:

$$\theta_{t+1,i} = \theta_{t,i}\beta^{m_{t,i}}$$

Note that for a binary classification problem, $m_{t,i}$ can be a 0-1 loss, namely 0 and 1 for correct and wrong predictions respectively. In this case, the update step of WM is simply a scaling of the weights: $\theta_{t+1,i} = \beta\theta_{t,i}$. Of course, the cost can also be a real value between, say, 0 and 1. It can be seen that WM adopts a multiplicative update strategy similar to EG, except that β does not have to be exp

[2]Different versions of the WM algorithm were proposed in 93, and the definitions are not consistent in literature. Other than the version we illustrate here, another commonly used version uses the update step $\theta_{t+1,i} = \beta\theta_{t,i}$ for expert i who made a wrong prediction. We use the former version here as it has a similar update to EG, although the base β is not required to be exp

After T rounds of game and using 0-1 loss, the WM algorithm has the following mistake upper bound:

$$M^T \leq 2(2-\beta)\min_i m_i^T + \frac{2\log d}{1-\beta}$$

where M^T and m_i^T are the number of mistakes that WM and the i^{th} expert make after T rounds respectively. It can be seen from this upper bound that the number of mistakes this algorithm makes will not be too far from the best expert (the gap between their mistakes does not increase over time).

- The Hedge Algorithm

The Hedge algorithm, proposed in 94 which is the same paper that proposed the highly celebrated AdaBoost algorithm, was originally used to update the weights assigned to weak learners (individual experts). Its update is similar to that of WM:

$$\theta_{t+1,i} = \theta_{t,i}\exp(-\lambda m_{t,i}) \tag{3.1}$$

However it differs from WM in that instead of making predictions by weighted majority vote, the Hedge makes predictions by randomized selections of experts. That is, at each round of the game the prediction is given by an expert sampled from the distribution $p_{t,i} = \theta_{t,i} / \sum_{j=1}^{d} \theta_{t,j}$. The multiplicative update pushes probability mass to those experts who frequently make correct decisions. Such as strategy is often used for online resource allocation (for example weak learner selection in AdaBoost).

This Hedge is very similar to the EG algorithm except that EG is applied in a general online convex optimization setting (just like the OGD algorithm), whereas the Hedge is applied in the "resource allocation" setting. Assuming the cost $m_{t,i} \in [-1, 1]$, the Hedge algorithm gives the following mistake upper bound (which is similar to that of the `prod` algorithm due to their close relationship):

$$\sum_{t=1}^{T} \mathbf{m}_t \cdot \mathbf{p}_t \leq \sum_{t=1}^{T} m_{t,i} + \lambda \sum_{t=1}^{T} (\mathbf{m}_t)^2 \cdot \mathbf{p}_t + \frac{\log d}{\lambda}$$

where $\mathbf{m}_t^2$ is the vector obtained by element-wise squaring the vector $\mathbf{m}_t$.

A major problem with the Hedge algorithm is that its performance is sensitive to the choice of learning rate λ, since the effect of a small perturbation in the learning rate gets exponentiated. Several extensions of Hedge exist that address this problem, for example 140 proposed a strategy that avoids tuning the learning rate manually, 141 modified the algorithm by treating Hedge as a dual-averaging scheme, 104, 142 proposed an adaptive learning rate for Hedge that is shown to be optimal in the worst case. Recent extensions of the algorithm can be found in 143.

- The `prod` Algorithm

The `prod` algorithm 95 has a similar setting as Hedge, but with a different update

strategy[3]:

$$\theta_{t+1,i} = \theta_{t,i}(1 - \lambda m_{t,i}), \quad \forall i = 1, \ldots, d \tag{3.2}$$

Compared with Eq. 3.1, this update replaces $\exp$ with a linear function as its first order approximation. Assuming that all costs $m_{t,i} \in [-1, 1]$, and let $\mathbf{p}_t = [p_{t,1}, \ldots, p_{t,d}]$, $p_{t,i} = \theta_{t,i} / \sum_{j=1}^{d} \theta_{t,j}$, $\mathbf{m}_t = [m_{t,1}, \ldots, m_{t,d}]$, then for $\lambda \leq 1/2$ we have the following mistake upper bound for the `prod` algorithm:

$$\sum_{t=1}^{T} \mathbf{m}_t \cdot \mathbf{p}_t \leq \sum_{t=1}^{T} m_{t,i} + \lambda \sum_{t=1}^{T} |m_{t,i}| + \frac{\log d}{\lambda}$$

3.3 Geometric Interpretations of EG and prod

As we have seen in Sec. 2.1.4, OMD is a general framework that derives many specific online learning algorithms. Therefore it is natural to ask if the multiplicative algorithms described in the previous section can also be derived from this framework. It is obvious that the Hedge algorithm, which is almost identical to EG, can be derived from OMD by letting the KL-divergence between $\boldsymbol{\theta}_{t+1}$ and $\boldsymbol{\theta}_t$. The WM algorithm can also be easily

[3]The update strategy adopted by `prod` appeared as early as in the work of Eggermont 144, though not applied and studied in an online learning setting. A follow-up work of 144 and detailed analysis of the `prod`-style update in applications of convex optimization with non-negativity constraints is given by 145.

derived from the OMD by using the generalized KL-divergence between $\boldsymbol{\theta}_{t+1}$ and $\boldsymbol{\theta}_t$:

$$KL_{\beta}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\theta}_t) = \sum_{i=1}^{d} \theta_{t+1,i} \log_{\beta} \frac{\theta_{t+1,i}}{\theta_{t,i}}$$

where $\log_{\beta}$ is the logarithm with base β. However, problem comes when we try to derive the `prob` algorithm from the OMD framework: we cannot find a convex function G whose corresponding OMD directly yields the `prob` update. That is to say, unlike many other online learning algorithms, the `prob` algorithm does not belong to the OMD framework, and the relationship between EG and `prob` has not been very clear. There is some work trying to interpret `prob` from different perspectives, for example in 134, `prob` is shown to be a special case of time-varying OMD. In this section, we give a different interpretation to the `prob` algorithm which also elucidates its relationship with EG.

3.3.1 Information Geometry and Mirror Descent

We discussed about information geometry and online learning in Chapter 2. Although the two fields were independently developed by different groups of researchers and seem irrelevant at first sight, it turns out that there is indeed a connection between them. In 146, it is shown that MD running on the primal manifold (i.e in the primal parameter space) is equivalent to NGD running on the dual manifold (i.e in the dual parameter space).

For each Bregman divergence D_G induced by a convex function G, there is a dual Bregman divergence D_H induced by a convex function H, which is the convex conjugate of

G: $H(\boldsymbol{\eta}) = \sup_{\boldsymbol{\theta}\in\boldsymbol{\Theta}}\{\langle \boldsymbol{\theta}, \boldsymbol{\eta}\rangle - G(\boldsymbol{\theta})\}$, thus D_G and D_H induce dual Riemannian manifolds with metrics $\nabla^2 G$ and $\nabla^2 H$ respectively. The dual coordinate systems are linked by $\boldsymbol{\eta} = g(\boldsymbol{\theta})$ and $\boldsymbol{\theta} = h(\boldsymbol{\eta})$, where $g = \nabla G$, $h = \nabla H$ satisfying $g = h^{-1}$ according to the properties of convex conjugate. The main conclusion given in 146 states that running MD with $\boldsymbol{\theta}$ is equivalent to running NGD with $\boldsymbol{\eta}$ (and vice verse). This actually can be easily seen from the OMD update:

$$g(\boldsymbol{\theta}_{t+1}) = g(\boldsymbol{\theta}_t) - \lambda_t \partial_{\boldsymbol{\theta}} \ell_{\boldsymbol{\theta}_t}$$

Here $g(\boldsymbol{\theta}_t)$ transforms $\boldsymbol{\theta}_t$ to the dual coordinate $\boldsymbol{\eta}_t$, and based on the relation $\boldsymbol{\theta} = h(\boldsymbol{\eta})$, $\partial_{\boldsymbol{\theta}} \ell_t$ indeed can be rewritten as $[\nabla^2 H]^{-1}_{\boldsymbol{\theta}} \partial_{\boldsymbol{\eta}} \ell_t$ via chain rule. That is to say, using the dual coordinate system the OMD update can be written as

$$\boldsymbol{\eta}_{t+1} = \boldsymbol{\eta}_t - \lambda_t [\nabla^2 H]^{-1}_{\boldsymbol{\theta}} \partial_{\boldsymbol{\eta}} \ell_{\boldsymbol{\eta}_t}$$

which is exactly the NGD running on the manifold with metric $\nabla^2 H$. Hence MD and NGD are essentially equivalent except that they use different coordinate systems, linked by the Fenchel transform. This relationship is summarized in Figure 3.1.

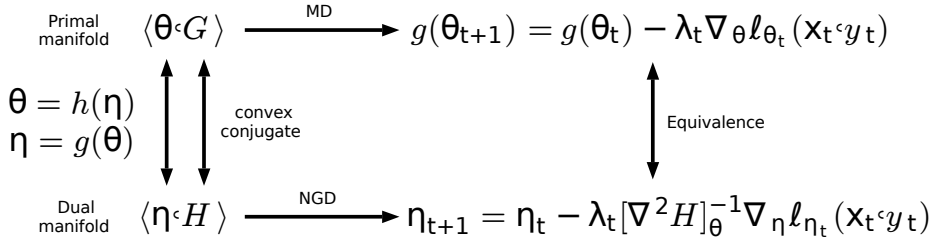


Figure 3.1: Relationship between MD and NGD.

3.3.2 Symmetry between EG and prod

Given the equivalence relationship between MD and NGD described in Sec. 3.3.1, we can run NGD on the primal manifold by running MD on the dual manifold first then projecting the solution back to the primal. As a trivial example, let $G(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|_2^2$ in which case the underlying manifold is Euclidean and $G = H$, thus running MD on the primal manifold and NGD on the dual leads to the same update, namely the normal gradient descent.

Now consider choosing $G(\boldsymbol{\theta}) = \sum_{i=1}^{d} \theta_i \log \theta_i$ and run MD on one manifold then project the update back to its dual. First, we examine running MD on the $\langle \boldsymbol{\theta}, G \rangle$ manifold then map it to the $\langle \boldsymbol{\eta}, H \rangle$ manifold. When running MD on the $\langle \boldsymbol{\theta}, G \rangle$ manifold, clearly this yields the EG algorithm as in Eq. 2.5. Since the coordinate map given by the Legendre transform is $\theta_i = h(\boldsymbol{\eta}) = \exp(\eta_i - 1)$ plug this equation into Eq.2.5 to map the udpate to the $\langle \boldsymbol{\eta}, H \rangle$ manifold, we have

$$\boldsymbol{\eta}_{t+1} = \boldsymbol{\eta}_t - \lambda_t \partial_{\boldsymbol{\eta}} \ell_{\boldsymbol{\eta}_t}$$

which is indeed gradient descent on the dual manifold.

Next, consider running MD on the $\langle \boldsymbol{\eta}, H \rangle$ manifold then map it back to the $\langle \boldsymbol{\theta}, G \rangle$ manifold. In this case, the convex conjugate $H(\boldsymbol{\eta}) = \sum_{i=1}^{d} \exp(\eta_i - 1)$, and let $\tilde{\lambda}_{t,i}$ be the specific learning rate for dimension i free to choose. The MD on the dual manifold is given

by

$$h(\boldsymbol{\eta}_{t+1})_i = h(\boldsymbol{\eta}_t)_i - \tilde{\lambda}_{t,i}\partial_{\eta_i}\grave{}_{\boldsymbol{\eta}_t}$$

$$\Rightarrow \quad \exp(\eta_{t+1,i} - 1) = \exp(\eta_{t,i} - 1) - \tilde{\lambda}_{t,i}\partial_{\eta_i}\grave{}_{\boldsymbol{\eta}_t}, \quad \forall i = 1, \ldots, d$$

Since we are free to determine $\tilde{\lambda}_{t,i}$, we choose $\tilde{\lambda}_{t,i} = \exp(\eta_{t,i} - 1)\lambda_t$. This way,

$$\exp(\eta_{t+1,i} - 1) = \exp(\eta_{t,i} - 1) - \exp(\eta_{t,i} - 1)\lambda_t\partial_{\eta_i}\grave{}_{\boldsymbol{\eta}_t}, \quad \forall i = 1, \ldots, d$$

$$\Rightarrow \quad \eta_{t+1,i} = \eta_{t,i} + \log(1 - \lambda_t\partial_{\eta_i}\grave{}_{\boldsymbol{\eta}_t}) \tag{3.3}$$

The coordinate map given by the Legendre transform is $\eta_i = g(\boldsymbol{\theta})_i = \log\theta_i + 1$, plug this equation into Eq.3.3 and map the update from the $(\boldsymbol{\eta}, H)$ manifold to $(\boldsymbol{\theta}, G)$, we have

$$\log\theta_{t+1,i} + 1 = \log\theta_{t,i} + 1 + \log(1 - \lambda_t\partial_{\eta_i}\grave{}_{\boldsymbol{\eta}_t})$$

$$\Rightarrow \quad \theta_{t+1,i} = \theta_{t,i}(1 - \lambda_t\partial_{\theta_i}\grave{}_{\boldsymbol{\theta}_t})$$

which yields the `prod` algorithm (in the last step we normalize $\theta_{t+1,i}$ so that it remains a d-simplex). The procedure described above is summarized in Table 3.1. It can be seen that the updates marked in red, given by the MD on the primal manifold and the NGD projected to the primal manifold, correspond exactly to the EG and `prod` algorithms. In other words, EG and `prod` are indeed symmetric algorithms running on dual manifolds! Therefore, the `prod` can also be treated as MD on the dual manifold, and we have set up a connection

	Primal	Dual
G, H	$G(\boldsymbol{\theta}) = \sum_{i=1}^{d} \theta_i \log \theta_i$	$H(\boldsymbol{\eta}) = \sum_{i=1}^{d} \exp(\eta_i - 1)$
g, h	$g(\boldsymbol{\theta}) = \log \theta_i + 1$	$h(\boldsymbol{\eta})_i = \exp(\eta_i - 1)$
MD	$\theta_{t+1,i} = \theta_{t,i} \exp\{-\lambda_t \partial_{\theta_i} \ell_t\}$	$\eta_{t+1} = \eta_t + \log(1 - \lambda_t \partial_{\eta_{t,i}} \ell_t)$
NGD (projected)	$\boldsymbol{\eta}_{t+1} = \boldsymbol{\eta}_t - \lambda_t \partial_{\boldsymbol{\eta}} \ell_{\boldsymbol{\eta}_t}$	$\theta_{t+1,i} = \theta_{t,i}(1 - \lambda_t \partial_{\theta_i} \ell_t)$

Table 3.1: MD and NGD for $G(\mathbf{t}) = \sum_{i=1}^{d} t_i \log t_i$. The updates marked in red (both on the primal manifold) correspond to the two types of multiplicative updates (EG and `prod`).

between EG and `prod` from a geometric perspective.

The interpretation given aboves coincides with the analysis given in 77. In that paper, the dual relationship between EG and `prod` (which had not even been officially proposed by that time, and was simply named "dual EG" algorithm) was also discovered, but from a different perspective.

3.4 Extensions of Multiplicative Algorithms

In this section, we give two extensions of the existing multiplicative algorithms, and show their applications in discriminative training tasks with contrast to additive algorithms. The first algorithm (Section 3.4.1) is a structured extension of the Winnow algorithm (Eq. 2.7), and the second algorithm (Section 3.4.2) is a multiplicative version of MIRA (Section 1.4.2).

3.4.1 The Structured Winnow Algorithm

3.4.1.1 The Main Algorithm

Given training data $T = \{(x_i, y_i)\}, i = 1, \ldots, |T|$, where x_i is the input (e.g. an Arabic sentence to translate) and y_i is the corresponding label (e.g. its English translation), and a baseline system to generate a set of hypotheses $\mathbf{GEN}(x_i)$ for each x_i, let each hypothesis $\hat{y} \in \mathbf{GEN}(x_i)$ be represented by $\mathbf{\Phi}(x_i, \hat{y}) = [\varphi_1(x_i, \hat{y}), \varphi_2(x_i, \hat{y}), \ldots, \varphi_d(x_i, \hat{y})]^T$, a d-dimensional feature vector. Let the score of each hypothesis be

$$\text{score}(\hat{y}) = \boldsymbol{\theta}^T \mathbf{\Phi}(x_i, \hat{y}) = \sum_{k=1}^{d} \theta_k \varphi_k(x_i, \hat{y}),$$

where $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_d]$ is the weight vector. Let $z \in \mathbf{GEN}(x_i)$ denote the hypothesis with the *highest model score*, and if $y_i \notin \mathbf{GEN}(x_i)$, replace it with the hypothesis that is closest to y_i according to some metric, i.e. the *oracle* hypothesis. The goal therefore is to learn a set of feature weights $\boldsymbol{\theta}$ such that $\text{score}(y_i) \geq \text{score}(z)$

The proposed training algorithm, shown in Figure 3.2, processes the training data sequentially, and makes an multiplicative update to the weight vector whenever an error is made ($z \neq y_i$). The algorithm iterates over the training set T for N passes (epochs). Note that for the Winnow algorithm $\sum_{k=1}^{d} \theta_k = 1$, namely the weight vector is always a probability distribution. Therefore, $\boldsymbol{\theta}$ is initialized to the uniform distribution, and after each update the new weight vector is normalized by the denominator Z_i.

Note also that a learning rate η must be specified by the user, and may significantly

Input: Training set $T = \{(x_i, y_i)\}$, $i = 1 \ldots |T|$
Initialization:
$\theta_{0,k} = 1/d$, $k = 1 \ldots d$
$t = 0$
Algorithm:
for $n = 1 \ldots N$ **do**
for $i = 1 \ldots |T|$ **do**
Predict $z = \underset{z \in \mathbf{GEN}(x_i)}{\operatorname{argmax}} \boldsymbol{\theta}^T \boldsymbol{\Phi}(x_i, z)$
if $z \neq y_i$ **then**
$\forall k \in [d]$
$\theta_{t+1,k} \leftarrow \frac{\theta_{t,k} \exp[\eta(\varphi_k(x_i, y_i) - \varphi_k(x_i, z))]}{Z_t}$
where
$Z_t = \sum_{k=1}^{d} \theta_{t,k} \exp\left[\eta(\varphi_k(x_i, y_i) - \varphi_k(x_i, z))\right]$
else
$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t$
end if
$t = t + 1$
end for
end for
return :

$$\begin{cases} \boldsymbol{\theta}_{N|T|} & \text{for non-averaged version} \\ \sum_{t=1}^{N|T|} \boldsymbol{\theta}_t \; / N|T| & \text{for averaged version} \end{cases}$$

Figure 3.2: Structured Winnow algorithm. The weight vector outputs are different for non-averaged and averaged version.

influence the performance of the algorithm, as discussed in Section 3.4.1.2.

Just as the averaged Perceptron 17 is an approximation to the voted Perceptron 147, a slight change to the structured Winnow algorithm, shown in the last line of Figure 3.2, yields an averaged Winnow that is empirically more stable and accurate.

3.4.1.2 Theoretical Analysis of the Algorithm

We show next that when the data is separable by a positive target weight vector $\mathbf{U}$, the algorithm of Figure 3.2 (non-averaged version) is guaranteed to converge. The cases where the target vector contains both positive and negative weights are discussed in Section 3.4.1.3.

Let $\overline{\mathbf{GEN}}(x_i) = \mathbf{GEN}(x_i) - \{y\}$ so that separability entails a hyper-plane between y_i and $\overline{\mathbf{GEN}}(x_i)$.

Theorem 1. *If there exists a vector* $\mathbf{U} \geq 0$ *such that* $\forall i, \forall z \in \overline{\mathbf{GEN}}(x_i), \mathbf{U}^T \mathbf{\Phi}(x_i, y_i) - \mathbf{U}^T \mathbf{\Phi}(x_i, z) \geq \delta$ *for some margin* $\delta > 0$*, then, for a suitable choice of* η*, the number of errors* M *that the Winnow algorithm of Figure 3.2 (non-averaged version) makes for any training sequence* $\{(x_i, y_i)\}$ *satisfies*

$$M \leq \frac{2 \log d \, R^2 \|\mathbf{U}\|_1^2}{\delta^2},$$

where $R = \max_i \max_{z \in \overline{\mathbf{GEN}}(x_i)} \|\mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, z)\|_\infty$.

Proof. Let $\mathbf{w} = \mathbf{U}/\|\mathbf{U}\|_1$, and since $\mathbf{U} \geq 0$, $\mathbf{w}$ is a distribution. Since the weight vector is explicitly normalized, $\boldsymbol{\theta}_t$ is also a distribution after each update t. Let $K_t = \sum_{k=1}^{d} w_k \log \frac{w_k}{\theta_{t,k}}$ be the K-L divergence between these two distributions. If an error occurs at a step t, $\boldsymbol{\theta}_t$ is updated to $\boldsymbol{\theta}_{t+1}$, so that

$$K_{t+1} - K_t = \sum_{k=1}^{d} w_k \log \frac{\theta_{t,k}}{\theta_{t+1,k}}$$

$$
\begin{aligned}
&= \sum_{k=1}^{d} w_k \log \frac{Z_t}{\exp[\eta(\varphi_k(x_t, y_t) - \varphi_k(x_t, z))]} \\
&= \log Z_t - \eta \sum_{k=1}^{d} w_k(\varphi_k(x_t, y_t) - \varphi_k(x_t, z)) \\
&\leq \log Z_t - \eta\delta/\|\mathbf{U}\|_1 \qquad (3.4)
\end{aligned}
$$

where the last inequality follows from the assumption that $\sum_{k=1}^{d} U_k[\varphi_k(x_t, y_t) - \varphi_k(x_t, z)] \geq \delta$.

Next, due to the convexity of $e^{\eta v}$,

$$e^{\eta v} \leq \frac{1+v}{2} e^{\eta} + \frac{1-v}{2} e^{-\eta} \qquad \forall\, v \in [-1, 1], \qquad (3.5)$$

and since $\|\mathbf{\Phi}(x_t, y_t) - \mathbf{\Phi}(x_t, z)\|_\infty \leq R$,

$$-R \leq \varphi_k(x_t, y_t) - \varphi_k(x_t, z) \leq R. \qquad (3.6)$$

Set $\frac{\varphi_k(x_t, y_t) - \varphi_k(x_t, z)}{R} = v$ in the definition of Z_t, to get

$$
\begin{aligned}
Z_t &= \sum_{d=1}^{k} \theta_{t,k} e^{\eta(\varphi_k(x_i, y_i) - \varphi_k(x_i, z))} \\
&\leq \sum_{k=1}^{d} \theta_{t,k} \left[\frac{1 + \frac{\varphi_k(x_t, y_t) - \varphi_k(x_t, z)}{R}}{2} e^{\eta R} \right. \\
&\quad \left. + \frac{1 - \frac{\varphi_k(x_t, y_t) - \varphi_k(x_t, z)}{R}}{2} e^{-\eta R} \right] \\
&= \frac{e^{\eta R} + e^{-\eta R}}{2} \sum_{k=1}^{d} \theta_{t,k} +
\end{aligned}
$$

$$\frac{e^{\eta R} - e^{-\eta R}}{2} \sum_{k=1}^{d} \theta_{t,k} \frac{\varphi_k(x_t, y_t) - \varphi_k(x_t, z)}{R}.$$

Now, $\sum_{k=1}^{d} \theta_{t,k} = 1$ and $\frac{e^{\eta R} - e^{-\eta R}}{2} > 0$, and if an error occurs, then $\sum_{k=1}^{d} \theta_{t,k}(\varphi_k(x_t, y_t) - \varphi_k(x_t, z)) \leq 0$. Therefore if an error occurs at update step t, then

$$\begin{aligned} Z_t &\leq \frac{e^{\eta R} + e^{-\eta R}}{2}, \quad \text{which leads to} \\ K_{t+1} - K_t &\leq \log \frac{e^{\eta R} + e^{-\eta R}}{2} - \eta \frac{\delta}{k\mathbf{U}k_1}. \end{aligned} \tag{3.7}$$

Invoking this recursively at each training error, we get

$$K_{t+1} \leq M \left(\log \frac{e^{\eta R} + e^{-\eta R}}{2} - \eta \delta / k\mathbf{U}k_1 \right) + K_1$$

Since $K_1 = \sum_{k=1}^{d} w_k \log(dw_k) \leq \sum_{k=1}^{d} w_k \log d = \log d$ and $0 \leq K_{t+1}$, it follows that

$$0 \geq M \left(\eta \frac{\delta}{k\mathbf{U}k_1} - \log \frac{e^{\eta R} + e^{-\eta R}}{2} \right) - \log d. \tag{3.8}$$

Setting $g(\eta) = \eta \frac{\delta}{k\mathbf{U}k_1} - \log \frac{e^{\eta R} + e^{-\eta R}}{2}$, it is easy to check that $g(\eta)$ is concave and its maximum is reached at

$$\eta^* = \frac{1}{2R} \log \frac{R + \delta k\mathbf{U}k_1}{R - \delta k\mathbf{U}k_1}. \tag{3.9}$$

Since $g(0) = 0$ and $g'(0) = \frac{\delta}{k\mathbf{U}k_1} > 0$, it must be true that $g(\eta^*) > 0$, and it follows from

(3.8) that

$$0 \geq M\,[g(\eta^*)] - \log d \implies M \leq \frac{\log d}{g(\eta^*)}. \tag{3.10}$$

Finally, setting

$$h(v) = \frac{1}{2}[(1+v)\log(1+v) + (1-v)\log(1-v)], \tag{3.11}$$

it is easy to check that $g(\eta^*) = h\left(\frac{\delta}{R\|\mathbf{U}\|_1}\right)$. Now, the function $\tilde{h}(v) = h(v) - \frac{v^2}{2}$ is convex with minimum 0, so that $h(v) \geq v^2/2$. Combining this (3.11) and (3.10)) yields

$$g(\eta^*) \geq \frac{\left(\frac{\delta}{R\|\mathbf{U}\|_1}\right)^2}{2} \implies M \leq \frac{2\log d\, R^2\|\mathbf{U}\|_1^2}{\delta^2},$$

as asserted. □

It follows from the proof that choice of the learning rate η is very important to the performance of the algorithm. If η is too large, $g(\eta)$ is smaller than 0, leading to unbounded training errors and the algorithm never converges. To see this, recall that $g(\eta)$ is concave, $g(0) = 0$ and $g'(0) > 0$, so $g(\cdot)$ must have another root $\eta_0 > 0$. Therefore, $g(\eta) > 0$ iff $\eta \in (0, \eta_0)$. In other words M is bounded only when $\eta \in (0, \eta_0)$. An analytical form of η_0 appears to be elusive, but we observe from (3.9) that

$$\delta = \log\frac{e^{\eta_0 R} + e^{-\eta_0 R}}{2} - \frac{\|\mathbf{U}\|_1}{\eta_0},$$

and it is easy to check that δ is a monotonically increasing function of η_0, indicating that if the margin is larger, we can select η from a wider interval.

3.4.1.3 Balanced Version of the Algorithm

The algorithm of Figure 3.2 maintains a positive weight vector at all times, assuming that the target vector is also positive. But what if the target vector contains negative or zero weights? To deal with this situation, we adopt the strategy of the $EG^{\pm}_{L}$ algorithm proposed in 82. We construct a new set of feature vectors

$$\boldsymbol{\Phi}'(x, y) \equiv [\boldsymbol{\Phi}(x, y) \mid -\boldsymbol{\Phi}(x, y)]^T$$

which is divided into positive and negative components of equal length. Then we can run the algorithm proposed in Section 3.4.1.1 as usual, but with the weight vector also divided into two parts, as

$$\boldsymbol{\theta}' = [\boldsymbol{\theta}^{+T} \mid \boldsymbol{\theta}^{-T}]^T$$

corresponding to $\boldsymbol{\Phi}(x, y)$ and $-\boldsymbol{\Phi}(x, y)$ respectively. Clearly,

$$\boldsymbol{\theta}'^T \boldsymbol{\Phi}'(x, y) = (\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-)^T \boldsymbol{\Phi}(x, y).$$

Therefore the actual weight vector is given by $\boldsymbol{\theta} = \boldsymbol{\theta}^+ - \boldsymbol{\theta}^-$ which may contain both positive and negative weights. This balanced version of structured Winnow is detailed in

Figure 3.3.

Since the spirit of the balanced algorithm is the same as the original version, the analysis to this algorithm is the same as in Section 3.4.1.2, except that since the balanced version is actually running on the extended data $\mathbf{\Phi}^a(x, y)$, the dimension becomes $2d$, and the target vector $\mathbf{U}$ and margin δ correspond to the new data as well.

3.4.1.4 Winnow and Perceptron: A Comparison

So why are we interested to use structured Winnow algorithm for discriminative training? It can be seen that the update of the Winnow algorithm is similar to Perceptron, except that the Winnow update is multiplicative(exponentiated) whereas the Perceptron update is additive. Each of these updating strategy has its advantage and disadvantage, which have been given detailed analysis in 82, 87. To make a comparison between the two algorithms, we quote the theorem for the error-bound on Perceptron from 17 as below(without making the assumption that $\|\mathbf{U}\|_2 = 1$):

Theorem 2 (Collins, 2002). *Let* $\overline{\mathbf{GEN}}(x_i) = \mathbf{GEN}(x_i) - \{y_i\}$, *and suppose there exists some vector* $\mathbf{U}$ *such that* $\forall i, \forall z \in \overline{\mathbf{GEN}}(x_i), \mathbf{U}^T\mathbf{\Phi}(x_i, y_i) - \mathbf{U}^T\mathbf{\Phi}(x_i, z) \geq \delta$, *and let* M *be the number of errors the Perceptron algorithm given by 17 make for any training sequence* (x_i, y_i), *then*

$$M \leq \frac{R^2\|\mathbf{U}\|_2^2}{\delta^2}$$

where $R = \max_{i, z \in \overline{\mathbf{GEN}}(x_i)} \|\mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, z)\|_2$

Compared with the Winnow error-bound (Theorem 1), it can be seen that Winnow uses l_1-norm for $\mathbf{U}$, l_∞ -norm for R whereas Perceptron uses l_2-norm for both $\mathbf{U}$ and R. Therefore whose performance is better depends on whether $2 \log d R^2_\infty \|\mathbf{U}\|^2_1$ is greater than $R^2_2 \|\mathbf{U}\|^2_2$ or not(R_∞ and R_2 meaning the R defined by l_∞ -norm and l_2-norm respectively).

When the target weight vector is sparse(a large part of the weights are zero or close to zero) and the training instances are dense, then $\|U\|_1 \approx \|U\|_2$ but $R_\infty \quad R \quad _2$, hence it is very likely that $2 \log d R^2_\infty \|\mathbf{U}\|^2_1 < R^2_2 \|\mathbf{U}\|^2_2$ and in this case Winnow will outperform Perceptron. On the other hand, when the target weight vector is dense but the training instances are sparse, then it very likely that $\|U\|_1 \quad \|U\| \quad _2$, $R_\infty \approx R_2$ and $2 \log d R^2_\infty \|\mathbf{U}\|^2_1 > R^2_2 \|\mathbf{U}\|^2_2$. In this case Perceptron will do better than Winnow.

We give a concrete example similar to the one given in 82 to compare the performance between Winnow and Perceptron. Suppose the feature dimenison d is very high, but the actual ranking of a hypothesis relies only on $k \quad d$ features, so the target weight vector is very sparse and we suppose $\mathbf{U} = [1, \ldots, 1_{k+1}, \ldots _d]$ where $_i$ are close to zero. On the other hand suppose we have a set of feature vectors such that $\mathbf{\Phi}_i(x_i, y) - \mathbf{\Phi}_i(x_i, z) = \{1, -1^n\}$ then in this case $\|\mathbf{U}\|_1 \approx k$, $\|\mathbf{U}\|_2 \approx \sqrt{k}$, $R_\infty = 1$ and $R_2 = \sqrt{d}$. The error upper-bound given by Winnow is around $2 \log d k \delta^2$ and the one given by Perceptron is around kd/δ^2. Therefore as long as $k < d/(2 \log d)$ Winnow will outperform Perceptron.

To summarize, when we have a large number of features but only a small part of them is relevant, then Winnow is more preferrable than Perceptron. In NLP tasks, it is very common that people define millions of features to describe an object(string, text, parse tree

Input: Training set $T = \{(x_i, y_i)\}$, $i = 1 \ldots |T|$
Initialization:
$\theta^+_{0,k} = \theta^-_{0,k} = 1/(2d)$, $k = 1 \ldots d$
$j = 0$
Algorithm:
for $n = 1 \ldots N$ **do**
for $i = 1 \ldots |T|$ **do**
Predict
$z = \underset{z \in \mathbf{GEN}(x_i)}{\text{argmax}} (\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-)^T \boldsymbol{\Phi}(x_i, z)$
if $z \neq y_i$ **then**
$\forall k \in [d]$
$\theta^+_{j+1,k} \leftarrow \frac{\theta^+_{j,k} \exp[\eta(\varphi_k(x_i,y_i) - \varphi_k(x_i,z))]}{Z_j}$,
$\theta^-_{j+1,k} \leftarrow \frac{\theta^-_{j,k} \exp[-\eta(\varphi_k(x_i,y_i) - \varphi_k(x_i,z))]}{Z_j}$
where

$$Z_j = \sum_{k=1}^{d} \theta^+_{j,k} \exp\left[\eta(\varphi_k(x_i, y_i) - \varphi_k(x_i, z))\right] + \theta^-_{j,k} \exp\left[-\eta(\varphi_k(x_i, y_i) - \varphi_k(x_i, z))\right]$$

else
$\boldsymbol{\theta}^+_{j+1} \leftarrow \boldsymbol{\theta}^+_j$
$\boldsymbol{\theta}^-_{j+1} \leftarrow \boldsymbol{\theta}^-_j$
end if
$j = j + 1$
end for
end for
return :

$$\boldsymbol{\theta} = \boldsymbol{\theta}^+_{N|T|} - \boldsymbol{\theta}^-_{N|T|} \quad \text{for non-averaged}$$

$$\boldsymbol{\theta} = \sum_{j=1}^{N|T|} \left(\boldsymbol{\theta}^+_j - \boldsymbol{\theta}^-_j\right) / N|T| \quad \text{for averaged}$$

Figure 3.3: Balanced structured Winnow alrgorithm. The output vectors differ for non-averaged and averaged version.

etc.) but many of them are uninformative, therefore Winnow might be a more suitable training algorithm in these cases.

3.4.1.5 Parallelization of Structured Winnow

When a large amount of training data is available, a distributed training strategy is essential for efficient large-scale training. 148 proposed a parallelized training algorithm named "Iterative Parameter Mixing" for structured Perceptron. The advantage of their algorithm is that the error-bound on the structured Perceptron holds even after parallization. For the structured Winnow algorithm, we propose a similar strategy that also maintains the validity of the error-bound, which we describe below.

Given a training set T, we split the set into S shards $T_1, \ldots, T_S$ Each shard runs on a single CPU core so that S shards are trained by Winnow in parallel. One epoch is finished once all shards finish their own training. At the end of each epoch, we need to collect the weights given by the training on individual shards and merge them, and the merged weight vector serves as the initial vector for the next epoch training shared by all shards. In 148, the merged weight vector is the arithmetic mean of individual weight vectors $\boldsymbol{\theta}^{s,n}$ from each shard after the n^{th} epoch:

$$\boldsymbol{\theta}^{AVG,n} = \sum_{s=1}^{S} \mu_{s,n} \boldsymbol{\theta}^{s,n}$$

where $\mu_{s,n}$ is the weight of the s^{th} shard for epoch n, and $\sum_s \mu_{s,n} = 1$ However, as we will

see in Theorem 3, the parallized structured Winnow requires geometric mean for weight merging instead of arithmetic mean, and in order for the merged weight vector to remain a probability distribution, the merged vector needs to be normalized:

$$\forall k \in [d]: \theta_k^{AVG,n} = \frac{1}{Z_n} \prod_{s=1}^{S} (\theta_k^{s,n})^{\mu_{s,n}}$$

where $Z_n = \sum_{k=1}^{d} \prod_{s=1}^{S} (\theta_k^{s,n})^{\mu_{s,n}}$. Similar to 148, we choose $\mu_{s,n}$ so that it is proportional to the errors the s^{th} shard made during the n^{th} epoch (denoted by $k_{s,n}$), namely

$$\mu_{s,n} = \frac{k_{s,n}}{k_n}$$

where k_n is the total number of errors made during the n^{th} epoch. The parallized structured Winnow algorithm is given in Figure 3.4.

We now give the following theorem showing that by using the parallelized version of structured Winnow described above, the error-bound still holds.

Theorem 3. *Assume a training set T is separable by a margin δ. Let $k_{s,n}$ be the number of errors that occurred on shard i during the n^{th} epoch of training. For any N, by using the parallelized Winnow training algorithm given in Figure 3.4,*

$$\sum_{n=1}^{N} \sum_{s=1}^{S} \mu_{s,n} k_{s,n} \leq \frac{2 \log d \prod_{n=1}^{N} Z_n \quad R^2 \|\mathbf{U}\|_1}{\delta^2}$$

Input: Training set $T = \{(x_i, y_i)\}$, $i = 1 \ldots |T|$
Initialization:
Divide T into S shards $T_1, \ldots, T_S$
$\forall k \in [d]\ \theta_k^{AVG,0} = 1/d,\ k = 1 \ldots d$
Algorithm:
for $n = 1 \ldots N$ **do**
$\forall s \in [S]$
$\boldsymbol{\theta}^{s,n} = \text{OneEpochWinnow}(T_s, \boldsymbol{\theta}^{AVG,n-1})$
$\forall k \in [d]$
$\theta_k^{AVG,n} = \frac{1}{Z_n} \prod_{s=1}^{S} (\theta_k^{s,n})^{\mu_{s,n}}$
where $Z_n = \sum_{k=1}^{d} \prod_{s=1}^{S} (\theta_k^{s,n})^{\mu_{s,n}}$
end for
return $\boldsymbol{\theta}^{AVG,N}$

Procedure OneEpochWinnow(T, $\boldsymbol{\theta}^{\prime}$)
$\boldsymbol{\theta} = \boldsymbol{\theta}^{\prime}$
for $i = 1 \ldots |T|$ **do**
Predict $z = \underset{z \in \mathbf{GEN}(x_i)}{\text{argmax}} \boldsymbol{\theta}^T \boldsymbol{\Phi}(x_i, z)$
if $z \neq y_i$ **then**
$\forall k \in [d]$
$\theta_{i+1,k} \leftarrow \frac{\theta_{i,k} \exp[\eta(\varphi_k(x_i, y_i) - \varphi_k(x_i, z))]}{Z_i}$
where
$Z_i = \sum_{k=1}^{d} \theta_{i,k} \exp\left[\eta(\varphi_k(x_i, y_i) - \varphi_k(x_i, z))\right]$
else
$\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i$
end if
end for
return $\boldsymbol{\theta}$

Figure 3.4: Parallelization of structured Winnow. The non-averaged Winnow is used here, for illustration, as the "OneEpochWinnow" procedure.

where $Z_n = \sum_{k=1}^{d} \prod_{s=1}^{S} (\theta_k^{s,n})^{\mu_{s,n}}$ *is the global normalization term after each epoch.*

Proof.

$$\begin{aligned}
&\sum_{s=1}^{S} \mu_{s,n} K_{s,n} = \sum_{s=1}^{S} \mu_{s,n} \sum_{k=1}^{d} w_k \log \frac{w_k}{\theta_k^{s,n}} \\
&= \sum_{k=1}^{d} \sum_{s=1}^{S} \mu_{s,n} \log \left(\frac{w_k}{\theta_k^{s,n}} \right)^{w_k} \\
&= \sum_{k=1}^{d} w_k \left(\sum_{s=1}^{S} \mu_{s,n} \log w_k - \sum_{s=1}^{S} \log (\theta_k^{s,n})^{\mu_{s,n}} \right) \\
&= \sum_{k=1}^{d} w_k \log \frac{w_k}{\prod_{s=1}^{S} (\theta_k^{s,n})^{\mu_{s,n}}} \\
&= \sum_{k=1}^{d} w_k \log \frac{w_k}{\theta_k^{AVG,n} Z_n} \\
&= K^{AVG,n} - \log Z_n
\end{aligned}$$

Notice that $\theta_k^{AVG,n-1}, \forall k$ are the initial weights for the n^{th} epoch, and by similar arguments in the proof of Theorem 1, we have $K^{AVG,n} - K^{AVG,n-1} \leq C k_{s,n}$ where $C = -(\frac{\delta}{R k \mathbf{u} k_1})^2/2$. Therefore

$$\begin{aligned}
&K^{AVG,n} \leq \sum_{s=1}^{S} \mu_{s,n} (C k_{s,n} + K^{AVG,n-1}) + \log Z_n \\
&\Rightarrow K^{AVG,n} - K^{AVG,n-1} \leq C \sum_{s=1}^{S} \mu_{s,n} k_{s,n} + \log Z_n \\
&\Rightarrow K^{AVG,N} - K^{AVG,0} \leq C \sum_{n=1}^{N} \sum_{s=1}^{S} \mu_{s,n} k_{s,n} + \sum_{n=1}^{N} \log Z_n
\end{aligned}$$

Since $K^{AVG,N} \geq 0$ and $K^{AVG,0} = \sum_{k=1}^{d} w_k \log(dw_k) \leq \sum_{k=1}^{d} w_k \log d = \log d$ we have

$$\sum_{n=1}^{N} \sum_{s=1}^{S} \mu_{s,n} k_{s,n} \leq (\log d - \sum_{n=1}^{N} \log Z_n)/(-C)$$
$$= \frac{2 \log(d / \prod_{n=1}^{N} Z_n) R^2 \|\mathbf{U}\|_1}{\delta^2}$$

□

Compared with the error upper-bound for serialized training(Theorem 1), it can be seen that there is an extra term $\prod_{n=1}^{N} Z_n$. Notice for $\forall k' \in [d], \forall s \in [S], \prod_{s=1}^{S} (\theta_{k'}^{s,n})^{\mu_{s,n}} = \theta_{k'}^{s',n} \prod_{s \neq s'} (\theta_{k'}^{s,n})^{\mu_{s,n}} \leq \theta_{k'}^{s',n}$, therefore $Z_n \leq \sum_{k=1}^{d} \theta_k^{s',n} = 1$. This means the error upper-bound for the parallelized Winnow can be higher than the serialized version, which is the side-effect of global normalization after each epoch. However as long as the training set T are evenly divided so that each shard T_s is very similar to each other, then for $\forall s$, $\theta_k^{s,n}$ and $\mu_{s,n}$ will be almost identical to each other, which indicates that the geometric mean of $\theta_k^{s,n}$ will be close to individual weights and $Z_n \approx 1$. Hence in practice, the side-effect of Z_n on the error upper-bound can be ignored by proper parallel training data preparation.

3.4.1.6 Experiments

In this section we show empirical results of discriminative training for parsing and statistical machine translation, using the proposed structured Winnow as well as Perceptron for comparison. Both tasks contain millions of features, and for both of them, we train the

discriminative models with two different settings. In the first setting we use only a small subset of the training data(compared with the number of features), and good solutions to this high-dimensional($p \gg N$) problem are expected to be sparse149. In the second setting the entire training set is used for model training, where denser solutions are more favorable.

Task 1: Parsing We used the Charniak parser37 for our experiment. We applied all the default settings of the parser, except that instead of using a MaxEnt reranker as the original paper did, we used the structured Perceptron and Winnow to rerank the 50-best parses. We used sections 2-21 of the Penn Treebank as the training set, section 24 as the held-out data to estimate the learning rate of the structured Winnow, and the entire section 23 as the evaluation set. There are ca 1.3 million features in all, and we selected the parse from a 50-best list with the highest F-score as the oracle.

In the first experimental setting, we used only about 1/5 of the entire training set [4] to train the discriminative model. In this setting, we chose $\eta = 0.3$ for the non-balanced Winnow and 0.1 for the balanced Winnow. Since the training set is not large, we ran Perceptron and Winnow in both serial and parallel modes(in the parallel mode, each section was treated as one shard). We trained the discriminative model for 50 epochs, and the performance on the evaluation set after each epoch is shown in Figure 3.5 and Figure 3.6, corresponding to serial and parallel training respectively. It can be seen clearly that Winnow outperformed Perceptron by a wide margin at any moment, and the balanced Winnow gave similar performance to the non-balanced version.

[4] In our experiment, we randomly selected sections 10, 14, 19, 21 as the small training set.

In the second experimental setting, we used the entire training set(20 sections) to train the discriminative model. In this setting, we chose $\eta = 0.25$ for the non-balanced Winnow and 0.1 for the balanced Winnow. Since the training set is much lager, we trained the discriminative model only in parallel mode. The performance on the evaluation set is shown in Figure 3.7. In this case, although the Winnow performed better than Perceptron in the first 10 epoch, the advantage vanished afterwards and eventually both algorithms gave similar results. The balanced Winnow in this case, however, did not behave as well as the other two.

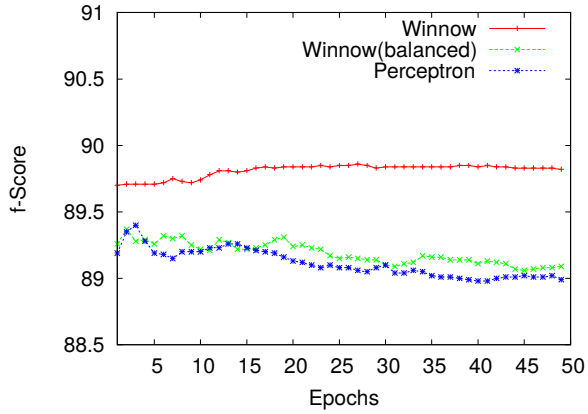


Figure 3.5: F-scores on the evaluation set after each epoch of *serial* training on 4 sections of the Penn Treebank.

Task2: Statistical Machine Translation We next show empirical results for an Arabic-English machine translation task. We used the direct translation model (DTM)150 as our translation system. The training corpus is the NIST Arabic-English parallel corpus for the GALE evaluation, which consists of ca 5 million sentence-pairs. We defined 23 million binary discrminative features including lexical features, lexical context features, Arabic

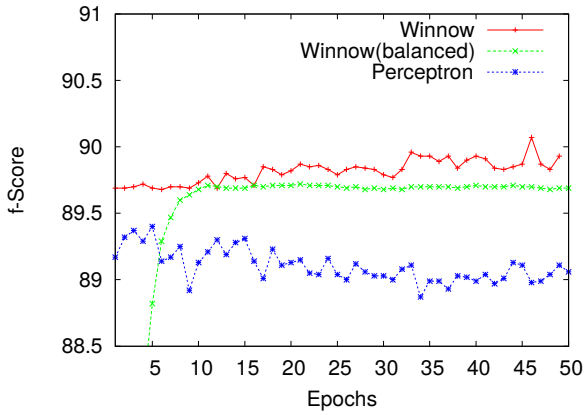


Figure 3.6: F-scores on the evaluation set after each epoch of *parallel* training on 4 sections of the Penn Treebank.

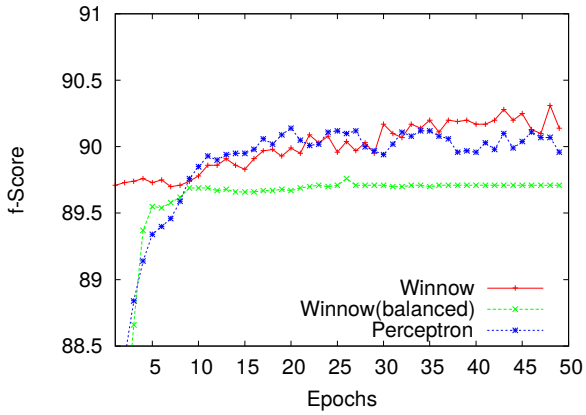


Figure 3.7: F-scores on the evaluation set after each epoch of parallel training on 20 sections of the Penn Treebank.

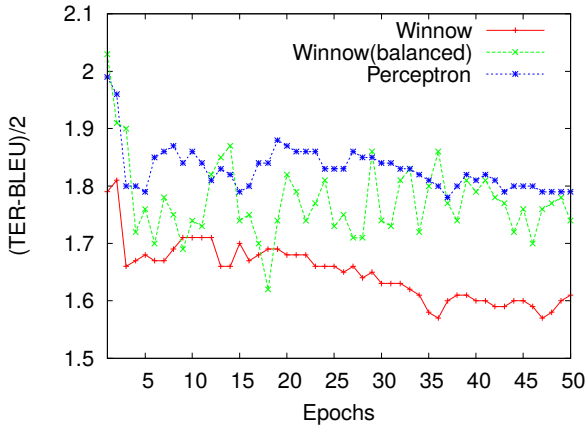


Figure 3.8: (TER-BLEU)/2 scores on the evaluation set after each epoch of *serial* training on 43,392 sentences.

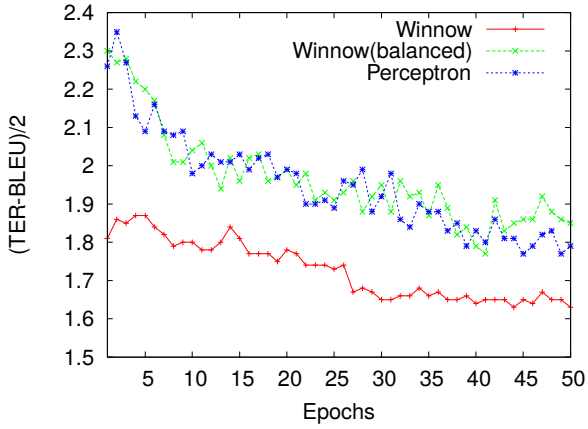


Figure 3.9: (TER-BLEU)/2 scores on the evaluation set after each epoch of *parallel* training on 43,392 sentences.

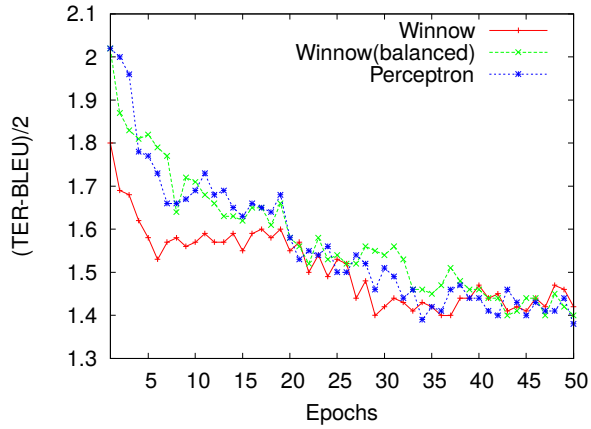


Figure 3.10: (TER-BLEU)/2 scores on the evaluation set after each epoch of parallel training on 0.4 million sentences.

segmentation features, part-of-speech features and coverage features.

We split the corpus into two parts: the first part containing 0.4 million sentences which was used for discriminative training, and all the remaining sentences comprise the second part. The translation model was trained on the second part, and the first part was decoded to get the 200-best translations for discrminative training. Using a complementary training set to train the translation model prevents over-fitting to the decoded data. In our experiments, we used (TER-BLEU)/2 12,151 as the evaluation metric, and chose the hypothesis in a 200-best list with the lowest sentence-level (TER-BLEU)/2 score as the oracle. Our evaluation set is part 1 of the GALE-DEV10 Arabic weblog (WB) data(1058 sentences), and we used part 2 as the held-out set(968 sentences).

In the first experimental setting, we trained the discriminative model on a small subset of 43,392 sentences randomly sampled from the entire training set. Since this data set is not huge, we ran Winnow and Perceptron in both serial and parallel mode(in the parallel mode,

we split the data into 12 shards, each containing 3616 sentences). We chose $\eta = 0.05$ for Winnow and balanced Winnow. Figure 3.8 and 3.9 show the (TER-BLEU)/2 scores on the evaluation set after each epoch for serial and parallel modes respectively. It can be seen that Winnow did better than Perceptron at all epochs, and the balanced Winnow in this experiment gave performance similar to Perceptron.

In our second experimental setting we scaled the training data to 0.4 million sentences. With this many sentences we only ran parallelized version of Winnow and Perceptron. The data was split into 110 shards with each shard containing about 3700 sentences. We chose $\eta = 0.1$ and 0.03 for Winnow and balanced Winnow respectively. Figure 3.10 shows the result on the evaluation set at each epoch. It can be seen that the although Winnow did better than Perceptron in the first a few epochs, the advantage of Winnow vanished after 20 epochs. The balanced Winnow in this experiment still behaved similarly to the Perceptron.

We also make a comparison of the training time between serial and parallel training in Table 3.2. The numbers of the serial training with the entire training set are predicted from a smaller set, since we did not actually run serial training in this case. It can be seen that parallel training significantly accelerated the training process, making large-scale training efficient.

3.4.2 The Multiplicative MIRA Algorithm

We now propose the second extension of multiplicative algorithms, the Multiplicative-MIRA (M-MIRA) algorithm.

	Exp1	Exp2
Serial Winnow	70	415*
Parallel Winnow	26	22
Serial balanced Winnow	150	830*
Parallel balanced Winnow	55	41
Serial Perceptron	43	277*
Parallel Perceptron	16	14

Table 3.2: A comparison of training time between serial and parallel training for 50 epochs in the machine translation task. Experiment 1 uses 43,392 sentences, experiment 2 uses 0.4 million sentences. The time unit is *core-hours*. The numbers with (*) are the predicted time.

3.4.2.1 The Algorithm

Our M-MIRA weight updating strategy is derived from the following problem formulation:

$$\begin{aligned} & \min_{\boldsymbol{\theta} \in \mathbb{R}^d} KL(\boldsymbol{\theta}, \boldsymbol{\theta}_t) + C\xi \\ s.t. \quad & \grave{}(\boldsymbol{\theta}, (x_t, y_t)) \leq \xi, \xi \geq 0 \\ & \boldsymbol{\theta} > 0, \sum_{i=1}^{d} \theta_i = 1 \end{aligned} \tag{3.12}$$

where KL refers to the Kullback-Leibler divergence between two distributions.

It can be seen that similar to MIRA (Section 1.4.2), M-MIRA also follows the "passive-aggressive" principle. However it differs from MIRA in two ways:

1. Weight vectors are forced to be probability distributions.

2. Entropy regularization is imposed.

The solution to the problem (3.12) has the following form:

$$\theta_{t,i} = \frac{\theta_{t,i} \exp(\tau \Delta\varphi_i)}{Z_t}, \quad i = 1 \ldots d$$
$$Z_t = \sum_{j=1}^{d} \theta_{t,j} \exp(\tau \Delta\varphi_j)$$
$$\tau = \min\{C, \tau^*\}$$

where τ^* is the root of the equation[5]

$$m(\tau) \triangleq \frac{1}{Z_t} \sum_{i=1}^{d} \theta_{t,i} \exp(\tau \Delta\varphi_i) \Delta\varphi_{t,i} = \rho_t \tag{3.13}$$

This solution gives a multiplicative update of the weight vectors, as the name of the algorithm indicates. Details of the derivation are omitted here. Note that unlike MIRA, the step size τ of M-MIRA does not have a closed-form solution, and a numerical procedure (for example Newton-Raphson) is required to find the exact solution of τ. However, if the $\exp(\cdot)$ term is approximated by its second-order Taylor expansion, we are able to give an approximate but closed-form solution of τ as well.

The complete M-MIRA is given in Figure 3.11.

3.4.2.2 Theoretical Analysis

The M-MIRA algorithm is similar to the structured Winnow (Section 3.4.1) except that due to the large-margin requirement, the learning rates are adaptive. Therefore, the regret

[5]Due to the non-decreasing property of $m(\tau)$, when the root of Eq. 3.13 does not exist, τ^* is set to $+\infty$.

Initialization: $\theta_i = 1/d, \quad i = 1 \ldots d$
Algorithm:
  for $t = 1 \ldots T$ **do**
    Receive training sample pair (x_t, y_t)
    Predict $\hat{y}_t = \underset{y \in Y(x_t)}{\text{argmax}} (\boldsymbol{\theta}_t \cdot \boldsymbol{\Phi}(x_t, y) + \rho(y_t, y))$
    if $\hat{y}_t \neq y_t$ **then**
      $\theta_{t+1,i} = \theta_{t,i} \exp(\tau \Delta\phi_i)/Z_t, \quad i = 1 \ldots d$
      $Z_t = \sum_{j=1}^{d} \theta_{t,j} \exp(\tau \Delta\phi_j)$
      $\tau = \min\{C, \tau^*\}$, where τ^* is the root of Eq. 3.13
    else
      $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t$
    end if
  end for

Figure 3.11: The Multiplicative Margin Infused Relaxed Algorithm(M-MIRA)

bound of M-MIRA can be derived in a similar manner and we omit the details here. Not surprisingly, the bound is also controlled by the l_1-norm of the target weight vector and l_∞-norm of the feature vector, therefore compared with MIRA, M-MIRA also tends to induce sparse weight vectors.

3.4.2.3 Experiments

In this section we report empirical results for a WMT Spanish to English statistical machine translation task, using MIRA, M-MIRA, as well as PRO as learning algorithms for comparison.

We used the Moses152 open-source machine translation system as our experiment platform. We trained a phrase-based translation model from the Europarl v7 Spanish-English parallel corpus, and a 5-gram language model with Kneser-Ney smoothing was trained

from the English Gigaword corpus. We used the WMT news-test 2011 as the tuning set(3003 sentences), news-test 2012 as the evaluation set (3003 sentences) and news-test 2011 (2489 sentences) as the development set to find proper aggressiveness parameter (C) values. We used BLEU score12 as the translation quality metric.

The Moses system has already integrated an implementation of the batch MIRA tuning strategy25, and we used this as the MIRA baseline for algorithm comparison and adopted the same strategy for our M-MIRA tuning. We implemented the M-MIRA simply by modifying the weight updating module of the Moses MIRA while keeping all other parts intact. We used both dense and sparse features for our experimental purpose. The dense features are the basic decoder features including the phrase translation model scores, reordering model scores, language model scores, word penalty etc, and there are 14 such features in all. The sparse features are those already implemented in Moses, including the target word insertion feature, source word deletion feature, word translation feature and the phrase length feature. There are in all ca. 150K sparse features.

We applied all the default settings of the Moses phrase-based decoder in all our experiments, and the size of the n-best list was set to 300. For all algorithms, we ran 20 iterations in the tuning process(the tuning set was decoded with the updated weights after each iteration). For MIRA and M-MIRA, 60 epochs over the decoded tuning set were run during each iteration. We set C to 0.01 and 0.001 for MIRA and M-MIRA respectively, determined from the development set. In the M-MIRA experiment, we used both the ex-

act solutions[6] of the step size τ as well as the approximate but closed-form solutions by second-order approximation for comparison.

Our first experiment used only dense features for decoding. Since dense features are usually very strong and highly relevant to the translation quality, we expect M-MIRA not to be significantly advantageous over MIRA in this learning process. Figure 3.12 shows the BLEU scores after each iteration on both the tuning and evaluation sets. In this case, it can be seen that MIRA and M-MIRA gave very similar performance. What is more, for M-MIRA the approximate closed-form solutions of τ yielded almost identical outputs as the exact numerical solutions. On the other hand, PRO also performed similarly to the other two.

Our second experiment used both dense and sparse features. The feature set is huge now and it may contain many features that are weakly relevant to the performance of our translation system, and the tuning set is not large enough for reliable weight learning. In this case we favor a sparse target weight vector and expect M-MIRA to outperform MIRA. The BLEU scores after each iteration on both the tuning and evaluation sets are plotted in Figure 3.13. From the tuning set plot we observe a much more rapid increase on the BLEU scores given by MIRA and PRO, however when it comes to the evaluation set M-MIRA won the competition. This indicates that MIRA and PRO is suffering from a more serious over-fitting problem, whereas the sparsity-promoting property of M-MIRA enables it focus more on the relevant features and generalize better. On the other hand, the approximate

[6]We used the Newton-Raphson method for root finding in our experiments.

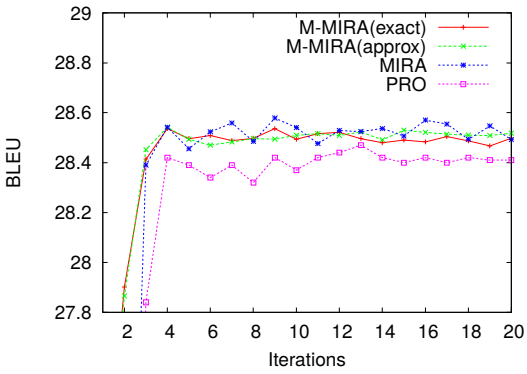
solutions of τ in this case did not behave equally well as the exact solutions.

The final scores given by all these algorithms on the evaluation set are given in Table 3.3. It can be seen that using MIRA as the learning algorithm, the extra sparse features did not help to improve the system using only dense features. Similar phenomenon was also reported in 153 for WMT Spanish-English translation task, who also used the Moses system, similar sparse features and MIRA for training, though their training sets are different. In contrast, M-MIRA(with exact root finding) lifted the score above the dense-feature baseline, therefore the potential power of these sparse features was better leveraged due to more effective learning.

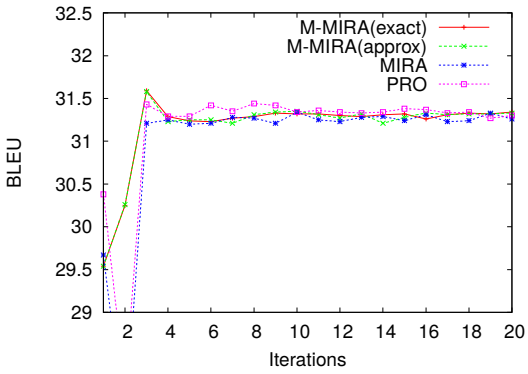
Algorithm	Dense	Dense+Sparse
PRO	31.30	31.31
MIRA	31.26	31.14
M-MIRA(approx)	31.33	31.30
M-MIRA(exact)	**31.34**	**31.47**

Table 3.3: BLEU scores on the evaluation set. Best scores are marked in boldsymbol.

To give more intuitive comparison between MIRA and M-MIRA, we also plot the actual weights of the first 3000 common features given by both algorithms in Figure 3.14. It can be seen that the solution given by M-MIRA is more sparse than MIRA, and the weights of the features that matter more in the learning process received more aggressive update.

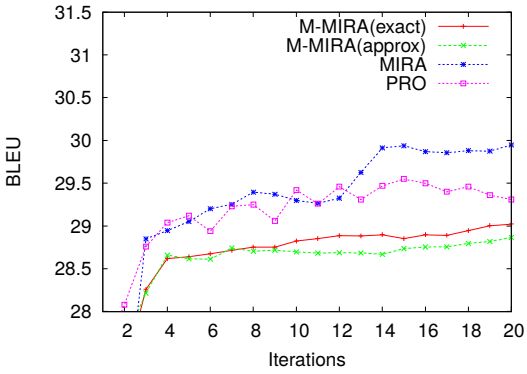


(a) Tuning set

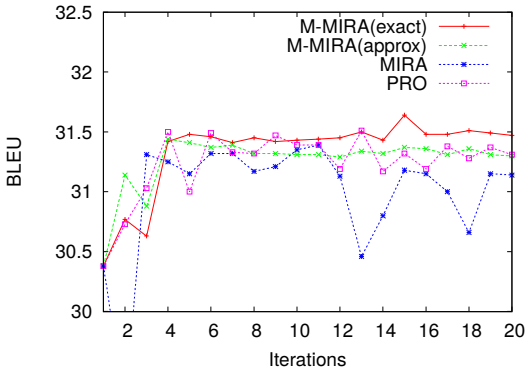


(b) Evaluation set

Figure 3.12: BLEU scores on the tuning and evaluation sets given by different algorithms after each iteration, using only dense features(for M-MIRA, "exact" and "approx" means using the exact and approximate solutions of τ respectively).



(a) Tuning set

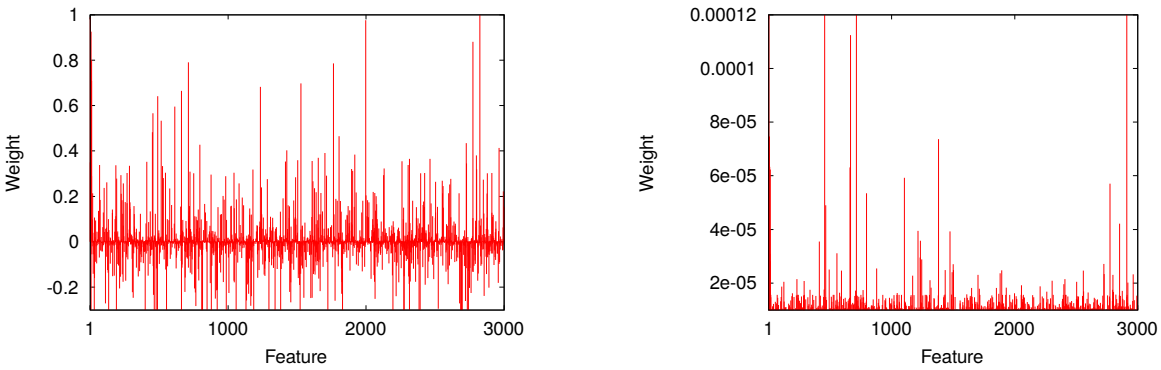


(b) Evaluation set

Figure 3.13: BLEU scores on the tuning and evaluation sets given by different algorithms after each iteration, using both dense and sparse features

3.5 Summary

In this chapter, we gave introduction, geometric interpretations and extensions to existing multiplicative online learning algorithms. The multiplicative algorithms are interesting as they often induce sparse solutions to the learning problem, which could be helpful in many NLP tasks for which millions of features are defined but only a small part of them



(a) MIRA weights **(b)** M-MIRA weights

Figure 3.14: Weights of the first 3000 features given by MIRA and M-MIRA.

are highly informative, as can be seen from our experimental results.

As discussed in Section 3.3, although OMD is a powerful framework that derives many specific online learning algorithms, it does not cover `prod`. A natural question to ask is if there is a different framework that subsumes both EG and `prod`, and derives other possible types of multiplicative algorithms? In Chapter 4, we propose a such a framework that is more suitable for deriving and analyzing multiplicative algorithms. Although we gave information geometric interpretation to the relationship between EG and `prod` in Section 3.3, their connections become even clearer under the new learning framework.

Chapter 4

Online Learning with General Divergences

4.1 Introduction

In this chapter we propose a new online learning framework. This framework, inspired by the study of information geometry, adopts the f-divergence as the proximity measure instead of Bregman divergence as in the OMD case. The resulting general online learning algorithm, named as the Generalized Multiplicative Update (GMU), is multiplicative in nature and can easily derive specific multiplicative algorithms. While EG and `prod` fail to be subsumed by the OMD framework at the same time, they are both special cases of GMU. We then propose a subclass of GMU, the q-Exponentiated Gradient (qEG) method, which further elucidates the relationship between EG, `prod` and other possible multiplicative

algorithms.

To understand the difference between OMD and GMU, we give analysis to these general algorithms from Riemannian geometric perspective. The analysis shows that they have different update strategies as they make different assumption about the manifold of the parameter space. Finally, motivated by the spirit of Nesterov's Accelerated Gradient Descent (AGD) algorithm, we also extend OMD and GMU to accelerated versions via adding momentum terms. The extensions not only recover AGD but also derive multiplicative versions of the accelerated algorithm.

4.2 Online Learning with f-Divergence

4.2.1 Learning Framework

Multiplicative algorithms form an interesting subclass of online learning algorithms. However, the classical OMD online learning framework is indeed not suitable for developing multiplicative algorithms, and it does not even cover the `prod` algorithm. In fact, it can be observed from the general OMD update rule (Eq. 2.2) that it is additive in nature. It is only when $g(\boldsymbol{\theta})$ is a $\log$ function that the additive update is converted to a multiplicative one, recovering the EG algorithm.

In this section, we consider the following learning framework, replacing Bregman di-

vergence with f-divergence as the proximity measure:

$$\boldsymbol{\theta}_{t+1} = \operatorname*{argmin}_{\boldsymbol{\theta}} \{\langle \boldsymbol{\theta}, \partial_t \rangle + \frac{1}{\lambda_t} D_f(\boldsymbol{\theta}_t, \boldsymbol{\theta})\} \tag{4.1}$$

where $D_f(\mathbf{p}, \mathbf{q})$ is the f-divergence induced by a convex function f (see Appendix C.1).

$$D_f(p, q) = \sum_{i=1}^{d} p_i f(\frac{q_i}{p_i})$$

The corresponding update rule is then derived as

$$\theta_{t+1,i} = \theta_{t,i} (f')^{-1}(-\lambda_t \partial_{\theta_i} \ell_t), \quad i = 1 \ldots d \tag{4.2}$$

It can be seen that compared with OMD, this framework is multiplicative in nature, and we call it the Generalized Multiplicative Update (GMU). Note that since $(\nabla f)^{-1} = \nabla f^*$ where f^* is the convex conjugate of f (see Appendix A.2), GMU can also be written as

$$\theta_{t+1,i} = \theta_{t,i} (f^*)'(-\lambda_t \partial_{\theta_i} \ell_t), \quad i = 1 \ldots d$$

The analysis of information geometry elucidates the relationship between the f and Bregman divergence, which can be summarized in Figure. 4.1. The f and Bregman divergences have a unique intersection, namely the α-divergence 125. When the divergences are defined on probability measure, then KL-divergence is the unique one belonging to

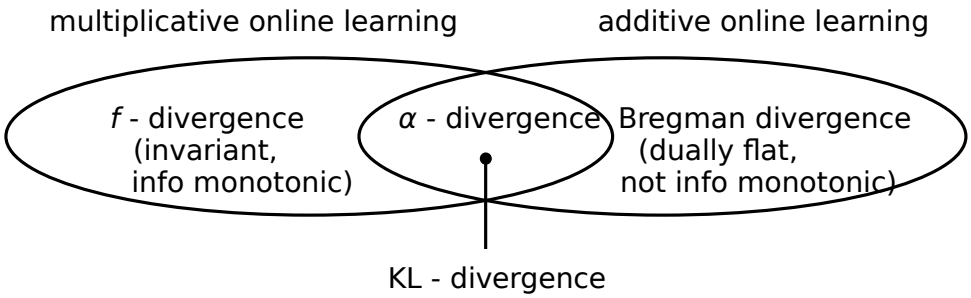


Figure 4.1: Relationship between f and Bregman divergences.

all f, Bregman and α divergences. This is why the OMD framework based on Bregman divergence is only able to derive the EG multiplicative algorithm: it is only when the KL-divergence is used that an intersection with GMU can be found which induces multiplicative algorithms.

We now show several examples of the algorithms derived from GMU with different choices of divergences in Table 4.1.

Table 4.1: Examples of algorithms derived from the proposed GMU.

$f(x)$	f-divergence	Algorithm
$\frac{1}{2}(x-1)^2$	χ^2-divergence	$\theta_{t+1,i} = \theta_{t,i}(1 - \lambda_t \partial_{\theta_i} \grave{}_t)$ (prod)
$x \log x$	KL-divergence	$\theta_{t+1,i} = \theta_{t,i} \exp\{-\lambda_t \partial_{\theta_i} \grave{}_t\}$ (EG)
$(\sqrt{x}-1)^2$	Hellinger distance	$\theta_{t+1,i} = \theta_{t,i}(1 + \lambda_t \partial_{\theta_i} \grave{}_t)^{-2}$ (new)

From the table we see that the `prod` algorithm, although not reachable by OMD, can be easily recovered from GMU corresponding to the χ^2-divergence. What is more, we have derived a new algorithm from the Hellinger distance, which we call the Reciprocal Squared Gradient (RSG) algorithm. Although the three algorithms (`prod`, EG, RSG) are derived from different divergences and seem irrelevant at first sight, they are in fact tightly related to each other, as we will see in the next section.

4.2.2 The q-Exponentiated Gradient Algorithm

In this section, we describe an interesting family of algorithms derived from the GMU framework based on deformed exponential.

4.2.2.1 Tsallis Statistics and the Deformed Exponential

In 1988, Constantino Tsallis proposed a generalization of the Shannon-Boltzmann-Gibbs entropy later known as the Tsallis entropy, which revolutionized statistical mechanics 154, 155. The Tsallis entropy of a distribution p is given by

$$S_q(p) = \frac{1 - \sum_i p_i^q}{q - 1} \tag{4.3}$$

where $q \in \mathbb{R}$ is a hyper-parameter, and when $q = 1$ the Tsallis entropy converges to the Shannon-Boltzmann-Gibbs entropy.

The Tsallis entropy can be equivalently written as

$$S_q(p) = -\sum_i p_i^q \log_q p_i$$

where $\log_q$ is called the q-logarithm 156, defined as

$$\log_q(x) = \frac{x^{1-q} - 1}{1 - q} \tag{4.4}$$

It can be shown that $\lim_{q \to 1} \log_q(x) = \log(x)$, therefore the q-logarithm is a generalization of

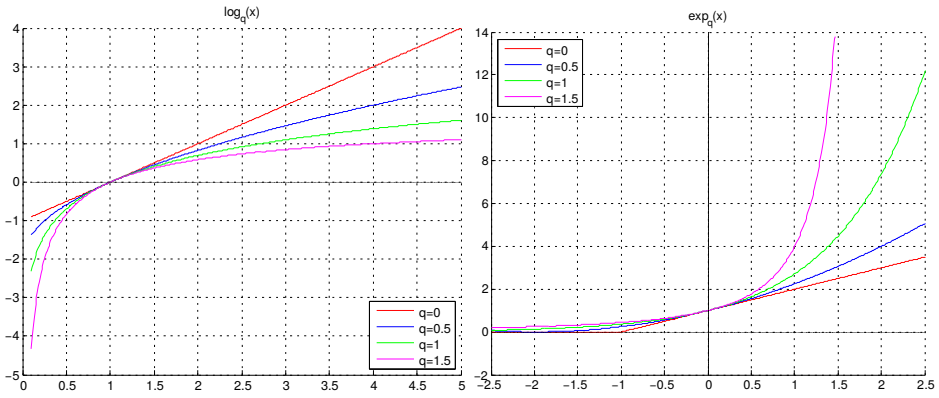


Figure 4.2: $\log_q$ and $\exp_q$ functions with different q values.

log The inverse of $\log_q$ is called the q-exponential, which is a generalization of the exp function and has the form

$$\exp_q(x) = [1 + (1 - q)x]_+^{\frac{1}{1-q}} \tag{4.5}$$

where $[x]_+ = x$ if $x > 0$ and 0 otherwise. The $\log_q$ and $\exp_q$ functions are plotted in Figure 4.2.

The q-logarithm and q-exponential satisfy many properties that generalizes the normal

log and exp functions, for example

$$\frac{d}{dx}\log_q(x) = x^{-q} \tag{4.6}$$

$$\frac{d}{dx}\exp_q(x) = \exp_q(x)^{q} = \exp_{2-\frac{1}{q}}(qx) \tag{4.7}$$

$$\log_q(1/x) = -\log_q(x) \tag{4.8}$$

$$\exp_q(x)\exp_q(-x) = 1 \tag{4.9}$$

$$\log_q(xy) = \log_q(x) + \log_q(y) + (1-q)\log_q(x)\log_q(y) \tag{4.10}$$

$$\exp_q(x)\exp_q(y) = \exp_q(x + y + (1-q)xy) \tag{4.11}$$

The last two properties are important, because it shows that unlike the normal log and exp for which $\log(xy) = \log(x) + \log(y)$, $\exp(xy) = \exp(x)\exp(y)$, the q-logarithm and q-exponential are in general *non-additive*. Therefore, the Tsallis entropy is also called non-additive entropy. A more comprehensive list of properties of the q-logarithm and exponential can be found in, for example, 157, 158.

The definition of q-logarithm and exponential lays the foundation of Tsallis statistics, and many related concepts and algorithms can also be generalized (for example the q-exponential family, q-divergences, q-logistic regression etc. 159, 160). In fact, the q-logarithm and exponential are special cases of even more general definitions of logarithm and exponential 161, 162, whose details we omit here.

4.2.2.2 The q-Exponentiated Gradient Algorithm

We now propose a family of multiplicative algorithms derived from GMU, by letting $(\nabla f)^{-1} = \exp_q$. The corresponding update has the form

$$\begin{aligned} \theta_{t+1,i} &= \theta_{t,i} \exp_q(-\lambda_t \partial_{\theta_i} \ell_t) \\ &= \theta_{t,i}[1 - (1-q)\lambda_t \partial_{\theta_i} \ell_t]_+^{\frac{1}{1-q}}, \quad i = 1 \ldots d \end{aligned} \tag{4.12}$$

and we call this algorithm the q-Exponentiated Gradient (qEG).

It can be easily verified that in this case

$$f(x) = \begin{cases} \frac{1}{1-q}\left(\frac{x^{2-q}}{2-q} - x\right) + C & q \neq 1, 2 \\ x \log x - x + 1 & q = 1 \\ -\log x + x - 1 & q = 2 \end{cases} \tag{4.13}$$

where $C = \frac{1}{2-q}$ to ensure $f(1) = 0$. The corresponding f-divergence can therefore also be determined:

$$\begin{aligned} D_f^{(q)}(a, b) &= \sum_i \frac{b_i(b_i^{1-q} - a_i^{1-q}(2-q))}{(2-q)(1-q)} + C \\ &= \frac{1}{(2-q)(q-1)}\left(1 - \sum_i a_i^{q-1} b_i^{2-q}\right) \end{aligned}$$

and $D_f^{(1)}(a, b) = KL(b, a)$ for $q = 1$, $D_f^{(2)}(a, b) = KL(b, a)$ for $q = 2$. This is just the

discrete version of the α-divergence (Eq. 2.15) with $\alpha = 3 - 2q$.[1] However, it should be noted that $f(x)$ given by Eq. 2.16 and 4.13 are different, and although both of them induce the same α-divergence (the correspondence between f and f-divergence is not necessarily one-to-one), the qEG algorithm (Eq. 4.12) cannot be derived by using Eq. 2.16.

By choosing different values of q (assuming $q \geq 0$), qEG can be specialized into different updates, and we give some examples in Table 4.2.

q	α-divergence	Algorithm
0	χ^2-divergence	$\theta_{t+1,i} = \theta_{t,i}(1 - \lambda_t \partial_{\theta_i} \ell_t)$ (prod)
0.5	(new)	$\theta_{t+1,i} = \theta_{t,i}(1 - \lambda_t \partial_{\theta_i} \ell_t)^2$ (new)
1	KL-divergence	$\theta_{t+1,i} = \theta_{t,i} \exp\{-\lambda_t \partial_{\theta_i} \ell_t\}$ (EG)
1.5	Hellinger distance	$\theta_{t+1,i} = \theta_{t,i}(1 + \lambda_t \partial_{\theta_i} \ell_t)^{-2}$ (RSG)

Table 4.2: Examples of qEG with different q values.

From this table it can be seen that although prod, EG and RSG appear to be irrelevant, their relationships can be easily discovered from the perspective of qEG: they are just special cases of the qEG algorithm with different q values! This is illustrated in Figure 4.3. It can be seen that the prod algorithm lies at the "boundary" of the qEG family ($q = 0$), the EG and RSG algorithms can be reached via smooth transition as the q increases to 1 and 1.5. When $q = 0.5$, we discover an algorithm that has a similar form of RSG and interpolates between prod and EG, though its corresponding α-divergence does not have an official name. It can also be seen that although χ^2-divergence, KL-divergence

[1] The α-divergence used here is often called the Amari α-divergence. In Appendix C.1.1, the Tsallis and Rényi α-divergences which have very similar forms are discussed. The Tsallis α-divergence can also be used to derive the qEG algorithm since it is just a scaled version of the Amari α-divergence. The Rényi α-divergences, however, cannot be directly applied since it does not admit closed form solutions in general.

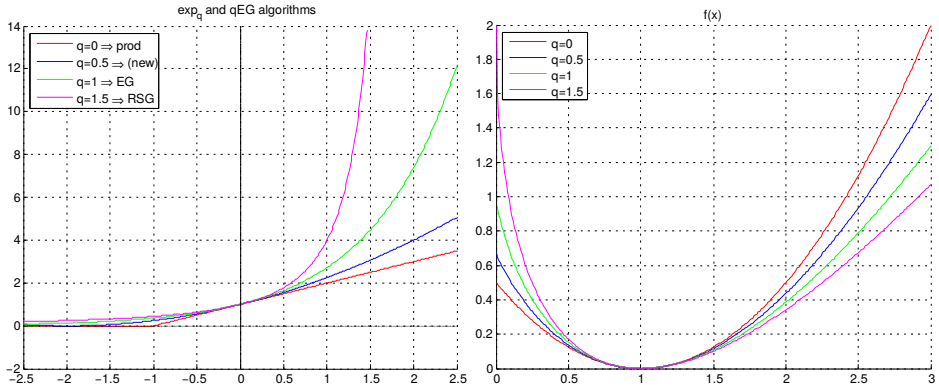


Figure 4.3: Left: From the perspective of qEG, the relationship between the prod, EG, and RSG algorithms are clear. Right: The corresponding $f(x)$ with different q values.

and Hellinger distance all belong to the general f-divergence, there is indeed a tighter connection between them: they are members of a smaller club, namely the α-divergence, with different α values.

4.2.2.3 Theoretical Analysis

In this section we give theoretical analysis to the qEG algorithm. Our analysis starts with the Tsallis divergence, which has a similar definition to the KL-divergence but using the q-logarithm 4.4 instead 163:

$$T\,D(p,\,q) \stackrel{\mathrm{X}}{=} p(x)\,\log_q \frac{p(x)}{q(x)}$$

In the anaylsis below we assume that the weights are normalized after each update, namely

$$\theta_{t+1,i} \;=\; \frac{1}{Z_t}\theta_{t,i}\exp_q(-\lambda_t \partial_{\theta_i} \grave{}_t)\,, \quad i = 1\ldots d$$

where $Z_t = \sum_{i=1}^{d} \theta_{t,i} \exp_q(-\lambda_t \partial_{\theta_i} \ell_t)$ is the potential function. Therefore the weights form a distribution after each update. We also consider a fixed learning rate $\lambda_t = \lambda$, and let the initial weights to be uniformly distributed $\theta_{1,i} = 1/d, \forall i = 1, \ldots, d$

Let $\boldsymbol{\theta}^*$ be the optimal weights in hindsight, and consider the following Tsallis divergence difference:

$$
\begin{aligned}
& T\,D(\boldsymbol{\theta}^*, \boldsymbol{\theta}_{t+1}) - T\,D(\boldsymbol{\theta}^*, \boldsymbol{\theta}_t) = \sum_{i=1}^{d} \theta_i^* \left(\log_q \frac{\theta_i^*}{\theta_{t+1,i}} - \log_q \frac{\theta_i^*}{\theta_{t,i}} \right) \\
&= \sum_{i=1}^{d} \theta_i^* \left(\log_q \frac{\theta_i^*}{\theta_{t+1,i}} + \log_q \frac{\theta_{t,i}}{\theta_i^*} \right) && (4.14) \\
&= \sum_{i=1}^{d} \theta_i^* \left(\log_q \frac{\theta_{t,i}}{\theta_{t+1,i}} - (1-q) A_{t,i} \right) && (4.15) \\
&= \sum_{i=1}^{d} \theta_i^* \left(\log_q \frac{Z_t}{\exp_q(-\lambda \partial_{t,i})} - (1-q) A_{t,i} \right) \\
&= \sum_{i=1}^{d} \theta_i^* \left(\log_q Z_t + \lambda \partial_{t,i} - (1-q) B_{t,i} - (1-q) A_{t,i} \right) && (4.16) \\
&= \log_q Z_t + \lambda \sum_{i=1}^{d} \theta_i^* \partial_{t,i} - (1-q) \sum_{i=1}^{d} \theta_i^* (A_{t,i} + B_{t,i}) && (4.17)
\end{aligned}
$$

where in Eq. 4.14, 4.15 and 4.16 we used the $\log_q$ properties Eq. 4.84.10, and $A_{t,i}$, $B_{t,i}$ are defined as

$$
\begin{aligned}
A_{t,i} &\triangleq \log_q \left(\frac{\theta_i^*}{\theta_{t+1,i}} \right) \log_q \left(\frac{\theta_{t,i}}{\theta_i^*} \right) \\
B_{t,i} &\triangleq (-\lambda \partial_{t,i}) \log_q Z_t
\end{aligned}
$$

On the other hand, it is easy to verify that for $0 \le q \le 2$ and $|x| \le 1$, $\exp_q(-x) \le 1 - x + \frac{q}{2}x^2$,[2] and that $\log_q(1-x) \le -x$. Therefore,

$$\begin{aligned}
\log_q Z_t &= \log_q \left(\sum_{i=1}^d \theta_{t,i} \exp_q(-\lambda \partial_{t,i}) \right) \\
&\le \log_q \left[\sum_{i=1}^d \theta_{t,i} \left(1 - (\lambda \partial_{t,i} - q\lambda^2 \partial_{t,i}^2) \right) \right] \\
&= \log_q \left[1 - \sum_{i=1}^d \theta_{t,i} (\lambda \partial_{t,i} - q\lambda^2 \partial_{t,i}^2) \right] \\
&\le \sum_{i=1}^d \theta_{t,i} (-\lambda \partial_{t,i} + q\lambda^2 \partial_{t,i}^2) \qquad (4.18)
\end{aligned}$$

Plug 4.18 into 4.17 and sum up $T\,D(\boldsymbol{\theta}^*, \boldsymbol{\theta}_{t+1}) - T\,D(\boldsymbol{\theta}^*, \boldsymbol{\theta}_t)$:

$$\begin{aligned}
\sum_{t=1}^T T\,D(\boldsymbol{\theta}^*, \boldsymbol{\theta}_{t+1}) - T\,D(\boldsymbol{\theta}^*, \boldsymbol{\theta}_t) &= T\,D(\boldsymbol{\theta}^*, \boldsymbol{\theta}_{T+1}) - T\,D(\boldsymbol{\theta}^*, \boldsymbol{\theta}_1) \\
&\le \sum_{t=1}^T \left[\sum_{i=1}^d \theta_{t,i} (-\lambda \partial_{t,i} + q\lambda^2 \partial_{t,i}^2) + \lambda \sum_{i=1}^d \theta_i^* \partial_{t,i} - (1-q) \sum_{i=1}^d \theta_i^* (A_{t,i} + B_{t,i}) \right]
\end{aligned}$$

[2] To see this, note via Taylor series expansion $\exp_q(-x) = 1 - x + \frac{q}{2}x^2 + \sum_{i \ge 3, i \in \text{odd}} -\frac{1}{i!} \prod_{j=1}^{i-1} (jq - (j-1)) x^i + \frac{1}{(i+1)!} \prod_{j=1}^{i} (jq - (j-1)) x^{i+1}$. For $0 \le q \le 2$, $-\frac{1}{i!} \prod_{j=1}^{i-1} (jq - (j-1)) > \frac{1}{(i+1)!} \prod_{j=1}^{i} (jq - (j-1))$. Therefore the inequality holds for $|x| \le 1$.

Rearranging the terms and note that $T\,D(\boldsymbol{\theta}^*, \boldsymbol{\theta}_{t+1}) \geq 0$, we have

$$\begin{aligned}
&\sum_{t=1}^{T}\left(\sum_{i=1}^{d}\theta_{t,i}\partial_{t,i} - \sum_{i=1}^{d}\theta_i^*\partial_{t,i}\right) \\
&\leq q\lambda\sum_{t=1}^{T}\sum_{i=1}^{d}\theta_{t,i}\partial_{t,i}^2 - \frac{(1-q)}{\lambda}\sum_{t=1}^{T}\sum_{i=1}^{d}\theta_i^*(A_{t,i}+B_{t,i}) + \frac{1}{\lambda}T\,D(\boldsymbol{\theta}^*, \boldsymbol{\theta}_1) \\
&\leq q\lambda\sum_{t=1}^{T}\sum_{i=1}^{d}\theta_{t,i}\partial_{t,i}^2 - \frac{(1-q)}{\lambda}\sum_{t=1}^{T}\sum_{i=1}^{d}\theta_i^*(A_{t,i}+B_{t,i}) + \frac{1}{\lambda}\log_q d \qquad (4.19)
\end{aligned}$$

where the last inquality follows from the fact that $T\,D(\boldsymbol{\theta}^*, \boldsymbol{\theta}_1) = \sum_{i=1}^{d}\theta_i^*\log_q\theta_i^* d \leq \sum_{i=1}^{d}\theta_i^*\log_q d = \log_q d$, thus we have given a regret bound for the qEG algorithm.

It can be seen that compared with the EG algorithm regret bound (Eq. 4.20), which we quote here:

$$R_T \leq \frac{\log(d)}{\lambda} + \frac{\lambda}{2}\sum_{t=1}^{T}\|\partial_t\|_\infty^2$$

Eq. 4.19 has an extra parameter q and term $-\frac{(1-q)}{\lambda}\sum_{t=1}^{T}\sum_{i=1}^{d}\theta_i^*(A_{t,i}+B_{t,i})$[3]. Therefore the qEG bound is a generalization of EG and the actual floating range is controlled by the selection of q.

[3]Note that we may easily re-write the bound Eq. 4.19 with $\|\partial_t\|_\infty$ by noticing that $\sum_{i=1}^{d}\theta_{t,i}\partial_{t,i}^2 \leq \|\partial_t\|_\infty^2$.

4.2.3 Related Work

Although to the best of our knowledge, using f-divergence in the online learning setting has not been explicitly proposed before, we note that the application of f-divergence (under the name of ϕ-divergence) in solving constrained optimization problem $\min f(x) : x \in \mathbb{R}^n_+$ has been studied in early work by Teboulle 164,165 and Iusem 166,167. Teboulle proposed proximal algorithm with ϕ-divergence as the proximity measure (named Entropic-Proximal Method (EPM))in 164, and studied its convergence properties in 165. Iusem also also gave some convergence analysis of EPM 166 and extended the algorithm by linearing the objective function (which is named BMIG in 167). However, these work did not show the convergence rate of the algorithms and contrast their properties with mirror descent, nor did they analyze the property from geometric perspective. The simpler formuation given by qEG that subsumes many specific examples was not discovered either.

We also note that although not mentioned in Seciton 4.2.2.2, the qEG algorithm with $q = 2$ recovers the "explicit" update proposed in 144. It was also in this early paper that the `prod` style-update algorithm was already proposed, as mentioned in the footnote on Page 60.

4.3 Geometric Interpretations of OMD and GMU

We have seen that OMD and GMU induce general additive and multiplicative updates respectively. In this section we give analysis to these algorithms from geometric perspec-

tives, and show that they make different assumptions about the manifold of the parameter space, which caused different algorithmic behaviors.

If we consider the underlying parameter space as a Riemannian manifold, the updated parameters $\boldsymbol{\theta}_{t+1}$ should be searched along a curve $\gamma(\lambda_t)$ with $\boldsymbol{\theta}_t$ being the starting point (namely $\gamma(0) = \boldsymbol{\theta}_t$) and some reasonable initial direction D 168. For example the direction can be given by the loss gradient, that is $D = \dot{\gamma}(0) = \partial_t$. In a Riemannian manifold, the geodesics satisfy the second order ODEs (see Appendix B.5)[4]:

$$\ddot{\gamma}^k(\lambda_t) + \dot{\gamma}^i(\lambda_t)\dot{\gamma}^j(\lambda_t)\Gamma_{ij}^k = 0, \quad \forall k = 1, \ldots, d \tag{4.20}$$

with initial conditions $\gamma(0) = \boldsymbol{\theta}_t$, $\dot{\gamma} = \partial_t$. Here Γ_{ij}^k are the connection coefficients (Christoffel symbols)[5] and usually we consider the Levi-Civita connection whose coefficients are given by

$$\Gamma_{ij}^k = \frac{1}{2} g^{kl} (\partial_j g_{li} + \partial_i g_{lj} - \partial_l g_{ij}) \tag{4.21}$$

where g_{ij} are the Riemannian metrics, and the corresponding geodesics are the shortest paths between two points. The parameter λ_t of the curve corresponds to the step size

[4]Note that we use the Einstein summation convention in this section.

[5]Strictly speaking Γ_{ij}^k should be written as $\Gamma_{ij}^k(\gamma(\lambda_t))$ since it is a function of points in the manifold. We write Γ_{ij}^k here instead for simplicity.

(learning rate) of the update, therefore the parameter update can be written as

$$\boldsymbol{\theta}_{t+1} = \gamma(\lambda_t)$$
$$\gamma(0) = \boldsymbol{\theta}_t, \dot{\gamma}(0) = \partial_t^{`}$$

From this perspective, as long as we specify the Riemannian metrics of the parameter manifold, the corresponding update can also be determined. A trivial example is the Euclidean case where the Riemannian metric is the identity matrix $M = diag[1, \ldots, 1]$ In this case $\Gamma_{ij}^k = 0, \forall i, j, k$ and the corresponding geodesic equation is simply $\ddot{\gamma}^i(\lambda_t) = 0, \forall i = 1, \ldots$ Therefore, $\boldsymbol{\theta}_{t+1} = \gamma(\lambda_t) = \boldsymbol{\theta}_t - \lambda_t \partial_t^{`}$ which recovers the gradient descent algorithm.

4.3.1 The Geometry of OMD

Consider a Riemannian metric given by

$$M = diag[\phi_1(\theta_1), \ldots, \phi_d(\theta_d)] \tag{4.22}$$

where φ_i are positive functions. Then the metric $g_{ii} = \varphi_i(\theta_i)$, $g^{ii} = 1/\varphi_i(\theta_i)$ and the Levi-Civita connection (Eq. 4.21) is given by

$$\Gamma_{ij}^k = \begin{cases} \frac{1}{2\varphi_i(\theta_i)} \frac{d\varphi_i(\theta_i)}{d\theta_i} & i = j = k \\ 0 & \text{otherwise} \end{cases}$$

The corresponding geodesic equation (Eq. 4.20) can thus be written as

$$\ddot{\gamma}^i + \frac{1}{2\varphi_i(\gamma^i)} \frac{d\varphi_i(\gamma^i)}{d\gamma^i} \left(\dot{\gamma}^i\right)^2 = 0, \quad \forall i = 1, \ldots, d \tag{4.23}$$

To solve this ODE, note that by multiplying $2\varphi_i(\gamma^i)\dot{\gamma}^i$ to both sides of Eq. 4.23, we get

$$2\varphi_i(\gamma^i)\dot{\gamma}^i\ddot{\gamma}^i + \varphi_i'(\gamma^i)\dot{\gamma}^{i\,3} = 0 \tag{4.24}$$

Note that $\frac{d}{\lambda_t}(\dot{\gamma}^{i\,2}) = 2\dot{\gamma}^i\ddot{\gamma}^i$ and $\frac{d}{dt}\varphi_i(\gamma^i) = \varphi_i'(\gamma^i)\dot{\gamma}^i$, Eq. 4.24 can be written as

$$\varphi_i(\gamma^i)\frac{d}{\lambda_t}(\dot{\gamma}^{i\,2}) + \frac{d}{dt}\varphi_i(\gamma^i)\dot{\gamma}^{i\,2} = \frac{d}{dt}\left(\dot{\gamma}^{i\,2}\varphi_i(\gamma^i)\right) = 0$$

Integrating $\frac{d}{dt}\left(\dot{\gamma}^{i\,2}\varphi_i(\gamma^i)\right)$ and notice the initial condition $\gamma^i(0) = \theta_{t,i}, \dot{\gamma}^i(0) = D_i$, we have

$$\dot{\gamma}^{i\,2}\varphi_i(\gamma^i) - D_i^2\varphi_i(\theta_{t,i}) = C$$

where C is a constant, and since the boundary condition (when $\gamma^j = \theta_{t,i}$) needs to hold, $C = 0$. Therefore $\dot{\gamma^j}^2 \varphi_i(\gamma^j) = D_i^2 \varphi_i(\theta_{t,i})$, or $\dot{\gamma^j} \varphi_i(\gamma^j)^{\frac{1}{2}} = D_i \varphi_i(\theta_{t,i})^{\frac{1}{2}}$. Hence,

$$\int_{\theta_{t,i}}^{\gamma^j} \varphi_i(\rho^j)^{\frac{1}{2}} d\rho = D_i \varphi_i(\theta_{t,i})^{\frac{1}{2}} \lambda_t$$

Let the antiderivative of $\varphi_i(\gamma)^{\frac{1}{2}}$ be $\Phi_i(\gamma)$, then $\Phi_i(\gamma^j) - \Phi_i(\theta_{t,i}) = D_i \varphi_i(\theta_{t,i})^{\frac{1}{2}} \lambda_t$, and the geodesic or the updated parameter can be written as

$$\theta_{t+1,i} = \gamma^j(\lambda_t) = \Phi_i^{-1} \left(D_i \varphi_i(\theta_{t,i})^{\frac{1}{2}} \lambda_t + \Phi_i(\theta_{t,i}) \right) , \quad \forall i = 1, \ldots, d \tag{4.25}$$

In general it is difficult to find a closed-form solution for Φ_i, however if we choose $\varphi_i(\theta) = \left(\frac{\partial^2 G}{\partial \theta_i^2} \right)^2$ where $G : \mathrm{R}^d \to \mathrm{R}$ is a convex function at least twice differentiable, and let $D_i = - \left(\frac{\partial^2 G}{\partial \theta_i^2} \right)^{-1} \partial_{\theta_i} \grave{}_t$, then Eq. 4.25 becomes

$$\theta_{t+1,i} = (G_i^0)^{-1} \left(-\lambda_t \partial_{\theta_i} \grave{}_t + G_i^0(\theta_{t,i}) \right) , \quad \forall i = 1, \ldots, d \tag{4.26}$$

or equilvalently

$$G_i^0(\theta_{t+1,i}) = G_i^0(\theta_{t,i}) - \lambda_t \partial_{\theta_i} \grave{}_t , \quad \forall i = 1, \ldots, d \tag{4.27}$$

where $G_i^0 , \ \frac{\partial G}{\partial \theta_i}$, and this recovers the OMD update Eq. 2.3.

From the analysis above it can be seen that key geometric assumption of OMD is that

the parameter manifold is intrinsically *flat*, since its geodesic (Eq.4.26) is linear after a coordinate transformation performed by $G_i^0(\theta_i)$. Another way to see the flatness of the manifold is that OMD in fact introduces a coordinate transformation whose corresponding Riemannian metric is the identity matrix. Note $\varphi_i(\theta_i) = \left(\frac{\partial^2 G}{\partial \theta_i^2}\right)^2$ and let $\eta_i = G_i^0(\theta_i)$, we have

$$
\begin{aligned}
& d\boldsymbol{\theta}^T M d\boldsymbol{\theta} \\
&= d\boldsymbol{\theta}^T diag\left(\frac{\partial^2 G}{\partial \theta_1^2}^2, \ldots, \frac{\partial^2 G}{\partial \theta_d^2}^2\right)^T diag\left(\frac{\partial^2 G}{\partial \theta_1^2}^2, \ldots, \frac{\partial^2 G}{\partial \theta_d^2}^2\right) d\boldsymbol{\theta} \\
&= d\boldsymbol{\theta}^T diag\left(\frac{\partial \eta_1}{\partial \theta_1}, \ldots \frac{\partial \eta_d}{\partial \theta_d}\right)^T diag\left(\frac{\partial \eta_1}{\partial \theta_1}, \ldots \frac{\partial \eta_d}{\partial \theta_d}\right) d\boldsymbol{\theta} \\
&= d\boldsymbol{\eta}^T d\boldsymbol{\eta}
\end{aligned}
$$

Therefore the metric for the $\boldsymbol{\eta}$ coordinate system is the identity matrix, or the manifold is intrinsically flat. This is also true for the general choice of M given by Eq. 4.22.

Obviously this property also holds for the more general update Eq. 4.25, however OMD provides us with a principled way of defining the geometry in which the metric tensor is induced by a convex function, and the update can be nicely formulated as Eq. 2.2 with the help of Bregman divergence. The initial direction $D_i = -\left(\frac{\partial^2 G}{\partial \theta_i^2}\right)^{-1} \partial_{\theta_i} \ell_t$ we choose is in fact the gradient of the loss represented in the $\boldsymbol{\eta}$ coordinate system, this is because $\frac{\partial \ell(\boldsymbol{\eta})}{\partial \theta_i} = \frac{\partial \ell(\boldsymbol{\eta})}{\partial \eta_i}\frac{\partial \eta_i}{\partial \theta_i}$ and therefore $D_i = -\frac{\partial \ell(\boldsymbol{\eta})}{\partial \eta_i} = -\left(\frac{\partial \eta_i}{\partial \theta_i}\right)^{-1}\frac{\partial \ell(\boldsymbol{\eta})}{\partial \theta_i} = -\left(\frac{\partial^2 G}{\partial \theta_i^2}\right)^{-1} \partial_{\theta_i} \ell_t$.

From the update Eq.4.27 it can also be seen that OMD essentially performs gradient descent in a transformed coordinate system. The choice of coordinate system determines

the properties of corresponding OMD algorithms. A good exposition on the effect of coordinate transformation can be found in 168.

4.3.2 The Geometry of GMU

We assume again that the update of GMU is be derived from the geodesics connecting $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t+1}$. Then the geodesic corresponding to the GMU update (Eq. 4.2) is given by

$$\gamma^i(\lambda_t) = \theta_{t,i}(f^0)^{-1}(-\lambda_t \partial_{\theta_i} \grave{}_t) \,, \quad i = 1 \ldots d \tag{4.28}$$

Unlike the OMD case, this geodesic is obviously non-linear. In fact, we may rewrite the GMU update as

$$\log \theta_{t+1,i} = \log \theta_{t,i} + (\log \circ f^{-1})(-\lambda_t \partial \grave{}_t) \,, \quad i = 1 \ldots d \tag{4.29}$$

and it can be seen that the geodesic (after a $\log$ coordinate transformation) is linear only when $(f^0)^{-1} = \exp$ corresponding to the EG algorithm. Therefore, the geometric assumption of GMU is that the parameter manifold is in general *non-flat*.

The Riemannian metric of the GMU manifold no longer has the diagonal form as in Eq. 4.22 (otherwise the manifold would be flat), which makes it difficult to specify M. By matching Eq. 4.28 and Eq. 4.20, the best we know about the Christoffel symbols (and thus

the metrics) is that they satisfy the following equation:

$$\theta_t^k (L^k)^2 \; (f^0)^{-1\;\prime\prime}(\lambda_t L^k) + \theta_t^i L^i \theta_t^j L^j \; (f^0)^{-1\;\prime}(\lambda_t L^i) \; (f^0)^{-1\;\prime}(\lambda_t L^j) \Gamma_{ij}^k = 0$$

where $L^i = -\partial_{\theta_i} \ell_t$. It is in general hard to find a closed-form solution of Γ_{ij}^k, and this makes further analysis of the GMU geometry difficult. However, from Eq. 4.29 we may see that intuitively, what GMU does is to transform the coordinates by $\log$ first, then distort the "straight" geodesic $\log \theta_{t+1,i} = \log \theta_{t,i} - \lambda_t \partial_{\theta_i} \ell_t$ (corresponding to the EG algorithm) by the function $\log \circ (f^0)^{-1}$ which reflects the curvature of the manifold. Therefore, GMU can be in general regarded as a variation of the EG algorithm.

Although a direct analysis of GMU may seem difficult, the geomtric analysis of q-EG as one of its special cases is more tractable. Consider the Riemannian metric

$$M = diag\left(-q \log_1 \theta + \frac{q}{2\theta_1}, \ldots, -q \log_d \theta + \frac{q}{2\theta_d}\right) \tag{4.30}$$

where q is the hyperparameter in $\exp_q$. It can be easily verified that for $0 < \theta_i \le 1, \forall i = 1, \ldots, d$, $g_{ii} = -q \log_i \theta + \frac{q}{2\theta_i} > 0$. Given this metric, we compute the α-connection given

by Eq. 2.13 with $\alpha = -q$ as follows:

$$\begin{aligned} \Gamma^{\alpha}_{iii} &= \Gamma^{0}_{ij,k} + \frac{q}{2} T_{ijk} \\ &= \frac{1}{2}\partial g_{ii} + \frac{q}{2} E_{\theta_i}[(\partial_i \ell_p)^3] \\ &= -\frac{q}{\theta_i} \end{aligned} \tag{4.31}$$

and $\Gamma^{\alpha}_{ijk} = 0$ if i, j, k are not equal. The corresponding geodesic equation with respect to Γ^{α}_{ijk} is written as

$$\ddot{\gamma}^i - \frac{q}{\gamma^i}(\dot{\gamma}^i)^2 = 0, \quad \forall i = 1, \ldots, d \tag{4.32}$$

It is easy to verify that the geodesics $\gamma^i(\lambda_t) = \theta_{t,i} \exp_q(-\lambda_t \partial_{\theta_i} \ell_t)$, $i = 1 \ldots d$ is a solution to Eq. 4.32. That is to say, the update of qEG conincides with the geodesics in the manifold equipped with the metric Eq. 4.30 and α-connection Eq. 4.31. Therefore it is tightly connected to the geometry induced by the f-divergence.

4.3.3 Discussion

From the analysis above we see that OMD and GMU make different assumptions about the parameter manifold. The flat manifold assumption makes it easy to do geometric analysis of OMD since we may write M and Γ^k_{ij} explicitly, but such a property does not hold for GMU. No assumption, however, is superior to the other in all cases. They each derive

algorithms (additive or multiplicative) that may outperform the other in some situations.

An important point to notice is that the geometric structures studied by information geometry, including the Riemannian metric and dual connections induced by Bregman and f-divergences (see Section 2.2.2), do not yield the OMD and GMU updates. In other words, using the Bregman and f-divergences as proximity measures (Eq. 2.2 and 4.1) does not simultaneously imply the geometry being defined by the structures Eq.2.12–2.14 and 2.19–2.21. This is an interesting point and we give more details below.

Consider the metric induced by the Bregman divergence $g_{ij}^{(D_G)}(\boldsymbol{\theta}) = \frac{\partial^2 G(\boldsymbol{\theta})}{\partial\theta^i \partial\theta^j}$ (Eq. 2.19) first. Assuming G is decomposable, namely $G(\boldsymbol{\theta}) = \sum_i u(\theta_i)$ for some convex function u, then $g_{ij}^{(D_G)} = 0$ for $i \neq j$. According to Eq. 4.25, the corresponding update is given by

$$\theta_{t+1,i} = U^{-1}\left(D_i u'(\theta_{t,i})^{\frac{1}{2}}\lambda_t + U(\theta_{t,i})\right), \quad \forall i = 1, \ldots, d \tag{4.33}$$

where $U = \int (u'')^{\frac{1}{2}}$. It is in general difficult to find a closed-form solution for U, and Eq. 4.33 is not consistent with the OMD update. Therefore, using g_{ij} as studied by the information geometry does not yield the OMD algorithm. However, when a closed-form solution for U can be found, Eq. 4.33 also yields some interesting algorithms. For example, let G be the negative entropy, namely $u(\theta) = \sum_i \theta \log\theta$ and $U(\theta) = 2\theta^{\frac{1}{2}}$. Using $D_i = -\left(\frac{\partial^2 G}{\partial\theta_i^2}\right)^{-1}\partial_{\theta_i}\ell_t$, then the corresponding is given by

$$\theta_{t+1,i} = \left(\theta_{t,i}^{\frac{1}{2}} + D_i u'(\theta_{t,i})^{\frac{1}{2}}\lambda_t\right)^2 \tag{4.34}$$

This is also an interesting algorithm, and although it can also be derived from OMD, the corresponding convex function is given by $G(\boldsymbol{\theta}) = \sum_i \frac{2}{3}\theta_i^{\frac{3}{2}} (\theta_i \geq 0)$ instead of the negative entropy.

We now take a look at the f-divergence case. In the GMU formulation, the parameters $\boldsymbol{\theta}$ are discrete distributions, and as discussed in Section 2.2.3 the Riemannian metric induced by f-divergence (Fisher information) for manifold of discrete distributions are given by $g_{ij} = \frac{1}{p_i}\delta_{i,j}$, regardless of the choice of f. In this case, the Riemannian metric is also consistent with Eq. 4.22, therefore the manifold of discrete distributions is also flat and the corresponding update derived from the geodesic is equivalent to Eq. 4.34.

Note however that in all the analysis above, we used the Levi-Civita connection and the corresponding geodesics to derive updates. While this is a natural choice, other connections are also possible. For example we may choose the dual connections studied by information geometry to derive updates, however such choices rarely yield closed-form solution and we omit the analysis here.

4.3.4 Related Work

There exist work that gives analysis to optimization algorithms (especially gradient descent) explicitly from the Riemannian geometric perspective 168–172. Unfortunately, not all of them got published. In 169 and its longer version 168, which inspired much analysis given in this section, GD and EG algorithms are treated as making different prior knowledge about the parameter distributions, which can be encoded as different Rieman-

nian metrics that reflects the prior knowledge. The general algorithms derived from the geometric analysis is very close to OMD (although this was not explicitly mentioned in the paper). Follow-up works 170–172[6] further extended the geometric analysis of algorithms to cases like spherical parameter space and geometry-sensitive loss functions etc.

4.4 Generalized Algorithms with Momentum

4.4.1 Nesterov's Accelerated Gradient Method

It is well-known that for first-order optimization methods, the optimal convergence rate ever reachable is $O(1/\sqrt{t})$[7] for non-smooth convex objectives and $O(1/t^2)$ for smooth convex objectives in which t is the number of iterations, see for example 173 (Section 3.2.1 and 2.1.2). In the non-smooth case where only subgradients can be used, the subgradient descent algorithm actually reaches the optimal upper-bound $O(1/\sqrt{t})$. By contrast, in the smooth case where gradients of the objective are available, the gradient descent algorithm enjoys convergence rate $O(1/t)$, which is not the optimal. Therefore, a natural question to ask is that if any better algorithms exist that reach the convergence rate upper-bound $O(1/t^2)$.

In 174, Yurii Nesterov, a living legend of optimization, proposed a first-order algorithm

[6] Interestingly, although rejected, the reviews of 171 can be found in `http://users.cecs.anu.edu.au/~williams/papers/P181reviews.txt`.

[7] That is $\ell(\boldsymbol{\theta}_t) - \ell^* \leq \frac{C}{\sqrt{t}}$ where ℓ^* is the minimum of the objective ℓ, and C a constant depending on the convexity of ℓ and initial point $\boldsymbol{\theta}_0$.

known as the accelerated gradient descent (AGD) that reaches the optimal convergence rate for smooth objectives. [8] In later years, Nesterov developed variations of the algorithm and extended its application to cases of non-smooth and composite objectives 175–177. Other researchers have also proposed extensions of the algorithm, for example the FISTA (Fast Iterative Shrinkage-Thresholding Algorithm) by 178.

The basic accelerated gradient descent algorithm is given as follows:

$$\boldsymbol{\eta}_0 = \boldsymbol{\theta}_0 = \boldsymbol{\theta}_{-1}$$

$$\boldsymbol{\eta}_t = \boldsymbol{\theta}_t + \frac{t-2}{t+1}(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) \tag{4.35}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\eta}_t - \lambda_t \nabla \ell(\boldsymbol{\eta}_t) \tag{4.36}$$

In the steps above, Eq. 4.36 performs just like a simple gradient descent, however instead of using $\boldsymbol{\theta}_{t+1}$ directly as the updated parameter, the actual updated parameter is given by Eq. 4.35 which carries momentum $\boldsymbol{\theta}_{t-1}$ from the previous iteration. The weight $\frac{t-2}{t+1}$ approaches to 1 as t gets larger, which accelerates convergence as the parameters are close to the optimum. Assuming ℓ is β-smooth and the step size $\lambda_t = \lambda < 1/\beta$ is fixed, it can be shown that the convergence rate of the algorithm is

$$\ell(\boldsymbol{\theta}_t) - \ell(\boldsymbol{\theta}^*) \leq \frac{2\|\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*\|}{\lambda(t+1)^2}$$

[8] The original paper was written in Russian and is hardly available. Nevertheless, many expositions on this topic exist, including Nesterov's own book 173. Possibly due to the same reason, the accelerated gradient algorithm did not receive enough attention until recent years.

Therefore, this rate is $O(1/t^2)$ which reaches the optimal for first-order methods.

4.4.2 MD with Momentum

In this section we show that mirror descent can also be extended by carrying a momentum. The extension, which we call Mirror Descent with Momentum (MDM), subsumes AGD as a special case, and a multiplicative version of AGD can be easily derived from this framework.

The MDM algorithm is formulated as follows:

$$\begin{aligned} \tilde{\boldsymbol{\theta}}_{t+1} &= \operatorname*{argmin}_{\boldsymbol{\theta}} \{ h\boldsymbol{\theta}, \alpha_t \partial`_t + \beta_t \partial`_{t-1} i + \gamma_t D_G(\boldsymbol{\theta}, \boldsymbol{\theta}_t) + \rho_t D_G(\boldsymbol{\theta}, \boldsymbol{\theta}_{t-1}) \} \quad (4.37) \\ \boldsymbol{\theta}_{t+1} &= \Pi_{\Omega}(\tilde{\boldsymbol{\theta}}_{t+1}) \end{aligned}$$

where α_t, β_t, γ_t, ρ_t are coefficients for each term at time t, and $\gamma_t + \rho_t = 1$ for normalization. It can be seen that compared with MD, the MDM considers loss gradients from history and uses double proximity measures.[9] The solution to Eq. 4.37 is given by

$$\tilde{\boldsymbol{\theta}}_{t+1} = \nabla G^{-1} [\gamma_t \nabla G(\boldsymbol{\theta}_t) + \rho_t \nabla G(\boldsymbol{\theta}_{t-1}) - \alpha_t \partial`_t - \beta_t \partial`_{t-1}] \quad (4.38)$$

We are now able to recover the AGD algorithm from the MDM framework. To see this,

[9] A recent work 68 also considers linear combination of loss functions from previous rounds, however their problem settings and algorithms are different from MDM.

note that by substituting Eq. 4.36 into Eq. 4.35, the AGD update becomes

$$\boldsymbol{\theta}_{t+1} = (1+\kappa)\boldsymbol{\theta}_t - \kappa\boldsymbol{\theta}_{t-1} - (1+\kappa)\lambda_t \partial \ell_t + \kappa\lambda_{t-1}\partial \ell_{t-1}$$

where $\kappa = \frac{t-2}{t+1}$. This is a special case of MDM where $G(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x}\|_2^2$, $\alpha_t = (1+\kappa)\lambda_t$, $\beta_t = -\kappa\lambda_{t-1}$, $\gamma_t = 1+\kappa$, $\rho_t = -\kappa$

As another example, consider $G(\boldsymbol{x}) = \sum_{i=1}^{d} x_i \log x_i$ where $\boldsymbol{x}$ is constrained to be a $(d-1)$-simplex, and choose α_t, β_t, γ_t, ρ_t as in the AGD case. The update 4.38 thus becomes

$$\tilde{\theta}_{t+1,i} = \theta_{t,i}^{1+\kappa}\theta_{t-1,i}^{-\kappa}\exp\left\{-(1+\kappa)\lambda_t \partial_{\theta_i}\ell_t + \kappa\lambda_{t-1}\partial_{\theta_i}\ell_{t-1}\right\}, \quad \forall i = 1, \ldots, d$$

followed by normalization of $\tilde{\boldsymbol{\theta}}_{t+1}$ so that they remain a probability simplex. This can be treated as an accelerated version of EG, and we call it AEG.

4.4.3 GMU with Momentum

In Section 4.2 we proposed GMU by replacing the Bregman divergence with f-divergence, it is therefore natural to ask whether GMU can be similarly extended to incorporate momentum terms just like MDM. An immediate thought would be to replace the Bregman divergences with f-divergences in Eq. 4.37 directly, however this is not feasible as no closed-form solutions would be permitted in general.

In the MDM formulation (Eq. 4.37), the sum $\gamma_t D_G(\boldsymbol{\theta}, \boldsymbol{\theta}_t) + \rho_t D_G(\boldsymbol{\theta}, \boldsymbol{\theta}_{t-1})$ can be

treated as a single divergence that measures the "distance" among three populations $\boldsymbol{\theta}$, $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t-1}$. In order to develop GMU with momentum, it is helpful to define a divergence similar to f-divergence but taking multiple entities into consideration. To this end, we define

$$\mathscr{D}_f(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3) = \sum_i \theta_{2,i}^{\gamma} \theta_{3,i}^{\rho} f\left(\frac{\theta_{1,i}}{\theta_{2,i}^{\gamma} \theta_{3,i}^{\rho}}\right) \tag{4.39}$$

where $\gamma, \rho \geq 0$ and $\gamma + \rho = 1$ is a generalized f-divergence measure. In fact, this extension can be treated as a special case of the f-dissimilarity 179 among k-populations

$$\mathbf{D}_{\mathbf{f}}(p_1, \ldots, p_k) = \sum_{x \in X} \mathbf{f}(p_1(x), \ldots, p_k(x))$$

where $\mathbf{f}$ is chosen to be $\theta_{2,i}^{\gamma} \theta_{3,i}^{\rho} f\left(\frac{\theta_{1,i}}{\theta_{2,i}^{\gamma} \theta_{3,i}^{\rho}}\right)$. Now we are able to formulate the problem of GMU with momentum as

$$\tilde{\boldsymbol{\theta}}_{t+1} = \operatorname*{argmin}_{\boldsymbol{\theta}} \{\langle \boldsymbol{\theta}, \alpha_t \partial \ell_t + \beta_t \partial \ell_{t-1} \rangle + \mathscr{D}_f(\boldsymbol{\theta}, \boldsymbol{\theta}_t, \boldsymbol{\theta}_{t-1})\}$$

whose update is given by

$$\tilde{\theta}_{t+1,i} = \theta_{t,i}^{\gamma_t} \theta_{t,i}^{\rho_t} (f')^{-1}\left(-\alpha_t \partial_{\theta_i} \ell_t - \beta_t \partial_{\theta_i} \ell_{t-1}\right), \quad \forall i = 1, \ldots, d$$

It can be seen that this enables a general multiplicative update carrying momentums from

previous iterations.

4.5 Summary

In this chapter, we proposed a new online learning framework (GMU) based on f-divergence. In contrast to OMD for which Bregman divergence is used as a proximity measure, GMU is specialized in deriving multiplicative online learning algorithms. While EG and `prod` cannot be subsumed in OMD at the same time, they can be easily derived from GMU. We also proposed a subclass of GMU, named qEG, by replacing the exponential used in EG with q-deformed exponential. From the perspective of qEG, the relationship between EG, `prod` as well as RSG (a new algorithm) becomes very clear: they are just special instances of qEG with different q values. We then gave interpretations to OMD and GMU from Riemannian geometric perspective: the flat manifold assumption by OMD and non-flat manifold assumption by GMU caused the difference in their updating strategies. Finally, we proposed accelerated versions of OMD and GMU by adding double proximity measures. The accelerated versions cover AGD as special cases, and new algorithms like AEG can also be derived.

Chapter 5

Training Conditional Random Fields with Natural Gradient Descent

5.1 Introduction

Graphical models are used extensively to solve NLP problems. One of the most popular models is the conditional random fields (CRF) 180, which has been successfully applied to tasks like shallow parsing 54, name entity recognition 181, word segmentation 182 etc., just to name a few.

While the modeling power demonstrated by CRF is critical for performance improvement, accurate parameter estimation of the model is equally important. As a general structured prediction problem, multiple training methods for CRF have been proposed corresponding to different choices of loss functions. For example, one of the common training

approach is maximum likelihood estimation (MLE), whose loss function is the (negative) log-likelihood of the training data and can be optimized by algorithms like L-BFGS 183, stochastic gradient descent (SGD) 75, stochastic meta-descent (SMD) 184 185 etc. If the structured hinge-loss is chosen instead, then the Passive-Aggressive (PA) algorithm 23, structured Perceptron 17 (corresponding to a hinge-loss with zero margin) etc. can be applied for learning.

In this chapter, we propose a novel CRF training procedure. Our loss functions are defined by the Bregman divergence between the model expectation and empirical mean of the feature vectors, and can be treated as a generalization of the log-likelihood loss. We then use natural gradient descent (NGD, see Section 2.2.4) to optimize the loss. Since for large-scale training, stochastic optimization is usually a better choice than batch optimization 72, we focus on the stochastic version of the algorithms. The proposed framework is very flexible, allowing us to choose proper convex functions inducing the Bregman divergences that leads to better training performances.

5.1.1 MLE for Graphical Models

Graphical models can be naturally viewed as exponential families 127. For example, for a data sample (x, y) where x is the input sequence and y is the label sequence, the conditional distribution $p(y|x)$ modeled by CRF can be written as

$$p_{\boldsymbol{\theta}}(y|x) = \exp\{\boldsymbol{\theta} \cdot \boldsymbol{\Phi}(x, y) - A_{\boldsymbol{\theta}}\}$$

where $\Phi(x, y) = \sum_{c \in C} \varphi_c(x_c, y_c) \in \mathbb{R}^d$ is the feature vector (of dimension d) collected from all factors C of the graph, and A_θ is the log-partition function.

MLE is commonly used to estimate the model parameters θ. The gradient of the log-likelihood of the training data is given by

$$\tilde{E} - E_\theta \tag{5.1}$$

where $\tilde{E}$ and E_θ denotes the empirical mean of and model expectation of the feature vectors respectively. The moment-matching condition $\tilde{E} = E_{\theta^*}$ holds when the maximum likelihood solution θ^* is found.

5.2 Algorithm

5.2.1 Loss Functions

The loss function defined for our training procedure is motivated by MLE. Since $E_\theta = \tilde{E}$ needs to hold when the solution to MLE is found, the "gap" between E_θ and $\tilde{E}$ (according to some measure) needs to be reduced as the training proceeds. We therefore define the Bregman divergence between E_θ and $\tilde{E}$ as our loss function. Since the Bregman divergence is in general asymmetric, we consider four types of loss functions as follows (named B_1-

B_4)[1]:

$$B_G\ \gamma E_{\theta}, \hat{E} - \rho E_{\theta} \qquad (B_1)$$

$$B_G\ \hat{E} - \rho E_{\theta}, \gamma E_{\theta} \qquad (B_2)$$

$$B_G\ \rho\hat{E}, E_{\theta} - \gamma\hat{E} \qquad (B_3)$$

$$B_G\ E_{\theta} - \gamma\hat{E}, \rho\hat{E} \qquad (B_4)$$

where $\gamma \in \mathbb{R}$ and $\rho > 1 - \gamma$. It can be seen that whenever the loss functions are minimized at point θ^* (Bregman divergence reaches zero), $E_{\theta^*} = \hat{E}$ and θ^* give the same solution as MLE.

We are free to choose the hyper-parameter γ which is possibly correlated with the performance of the algorithm. However to simplify the problem setting, we will only focus on the cases where $\gamma = 0$ or 1. Although it seems we now have eight versions of loss functions, it will be seen shortly that by properly choosing the convex function G, many of them are redundant and we end up having only two update strategies.

5.2.2 Applying NGD

The gradients of loss functions B_1-B_4 with respect to θ are given in Table 5.1.

[1] In the most general setting, we may use $B(aE_{\theta} - b\hat{E}, cE_{\theta} - d\hat{E})$ *s.t.* $a - b = c - d$, $a, b, c, d \in \mathbb{R}$ as the loss functions, which guarantees $E_{\theta} = \hat{E}$ holds at its minimum. However, this formulation brings too much design freedom which complicates the problem, since we are free to choose the parameters a, b, c, d as well as the convex function G. Therefore, we narrow down our scope to the four special cases of the loss functions $B_1 - B_4$ given above.

Loss	Gradient wrt. $\boldsymbol{\theta}$	γ
B_1	$\nabla_{\boldsymbol{\theta}}\mathrm{E}_{\boldsymbol{\theta}}\ \nabla G(\mathrm{E}_{\boldsymbol{\theta}}) - \nabla G(\hat{\mathrm{E}})$	1
	$\nabla_{\boldsymbol{\theta}}\mathrm{E}_{\boldsymbol{\theta}}\nabla^2 G(\hat{\mathrm{E}} - \mathrm{E}_{\boldsymbol{\theta}})(\mathrm{E}_{\boldsymbol{\theta}} - \hat{\mathrm{E}})$	0
B_2	$\nabla_{\boldsymbol{\theta}}\mathrm{E}_{\boldsymbol{\theta}}\nabla^2 G(\mathrm{E}_{\boldsymbol{\theta}})(\mathrm{E}_{\boldsymbol{\theta}} - \hat{\mathrm{E}})$	1
	$\nabla_{\boldsymbol{\theta}}\mathrm{E}_{\boldsymbol{\theta}}\ \nabla G(\mathbf{0}) - \nabla G(\hat{\mathrm{E}} - \mathrm{E}_{\boldsymbol{\theta}})$	0
B_3	$\nabla_{\boldsymbol{\theta}}\mathrm{E}_{\boldsymbol{\theta}}\nabla^2 G(\mathrm{E}_{\boldsymbol{\theta}} - \gamma\hat{\mathrm{E}})(\mathrm{E}_{\boldsymbol{\theta}} - \hat{\mathrm{E}})$	$\{0, 1\}$
B_4	$\nabla_{\boldsymbol{\theta}}\mathrm{E}_{\boldsymbol{\theta}}\ \nabla G(\mathrm{E}_{\boldsymbol{\theta}} - \gamma\hat{\mathrm{E}}) - \nabla G(\rho\hat{\mathrm{E}})$	$\{0, 1\}$

Table 5.1: Gradients of loss functions B_1-B_4.

It is in general difficult to compute the gradients of the loss functions, as all of them contain the term $\nabla_{\boldsymbol{\theta}}\mathrm{E}_{\boldsymbol{\theta}}$, whose computation is usually non-trivial. However, it turns out that the *natural gradients* of the loss functions can be handled easily. To see this, note that for distributions in exponential family, $\nabla_{\boldsymbol{\theta}}\mathrm{E}_{\boldsymbol{\theta}} = \nabla^2_{\boldsymbol{\theta}} A_{\boldsymbol{\theta}_t} = M_{\boldsymbol{\theta}_t}$, which is the Fisher information matrix. Therefore the NGD update (Eq. 2.23) becomes

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t \nabla_{\boldsymbol{\theta}}\mathrm{E}^{-1}_{\boldsymbol{\theta}_t}\nabla\ell(\boldsymbol{\theta}_t) \tag{5.2}$$

Now if we plug, for example $\nabla_{\boldsymbol{\theta}} B_1$, $\gamma = 0$, into Eq. 5.2, we have

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \lambda_t \nabla_{\boldsymbol{\theta}}\mathrm{E}^{-1}_{\boldsymbol{\theta}_t}\nabla_{\boldsymbol{\theta}} B_2 \\ &= \boldsymbol{\theta}_t - \lambda_t \nabla^2 G(\hat{\mathrm{E}} - \mathrm{E}_{\boldsymbol{\theta}_t})(\mathrm{E}_{\boldsymbol{\theta}_t} - \hat{\mathrm{E}})\end{aligned} \tag{5.3}$$

Thus the step of computing the the Fisher information can be circumvented, making the optimization of our loss functions tractable[2].

[2] In **?**, a similar trick was also used. However their technique was developed from a specific variational

This trick applies to all gradients in Table 5.1, yielding multiple update strategies. Note that $\nabla_{\boldsymbol{\theta}} B_1,\ \gamma = 1$ is equivalent to $\nabla_{\boldsymbol{\theta}} B_4,\ \gamma = 0$, and $\nabla_{\boldsymbol{\theta}} B_2,\ \gamma = 1$ is equivalent to $\nabla_{\boldsymbol{\theta}} B_3,\ \gamma = 0$. By applying Eq. 5.2 to all unique gradients, the following types of updates are derived (named U_1-U_6):

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t \nabla^2 G(\hat{\mathbf{E}} - \mathrm{E}_{\boldsymbol{\theta}_t})(\mathrm{E}_{\boldsymbol{\theta}_t} - \hat{\mathbf{E}}) \qquad (U_1)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t \left[\nabla G(\mathbf{0}) - \nabla G(\hat{\mathbf{E}} - \mathrm{E}_{\boldsymbol{\theta}_t}) \right] \qquad (U_2)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t \nabla^2 G(\mathrm{E}_{\boldsymbol{\theta}_t} - \hat{\mathbf{E}})(\mathrm{E}_{\boldsymbol{\theta}_t} - \hat{\mathbf{E}}) \qquad (U_3)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t \nabla^2 G(\mathrm{E}_{\boldsymbol{\theta}_t})(\mathrm{E}_{\boldsymbol{\theta}_t} - \hat{\mathbf{E}}) \qquad (U_4)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t \left[\nabla G(\mathrm{E}_{\boldsymbol{\theta}_t} - \hat{\mathbf{E}}) - \nabla G(\mathbf{0}) \right] \qquad (U_5)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t \left[\nabla G(\mathrm{E}_{\boldsymbol{\theta}_t}) - \nabla G(\hat{\mathbf{E}}) \right] \qquad (U_6)$$

5.2.3 Reducing the Types of Updates

Although the framework described so far is very flexible and multiple update strategies can be derived, it is not a good idea to try them all in turn for a given task. Therefore, it is necessary to reduce the types of updates and simplify the problem.

We first remove U_4 and U_6 from the list, since they can be recovered from U_3 and U_5 respectively by choosing $G^q(\mathbf{u}) = G(\mathbf{u} + \hat{\mathbf{E}})$. To further reduce the update types, we impose the following constraints on the convex function G:

---

inference problem setting, whereas the proposed approach derived from Bregman divergence is more general.

1. G is symmetric: $G(\mathbf{u}) = G(-\mathbf{u})$.

2. $\nabla G(\mathbf{u})$ is an element-wise function, namely $\nabla G(\mathbf{u})_i = g(u_i)$, $\forall i \in 1, \ldots, d$, where g_i is a uni-variate scalar function.

For example, $G(\mathbf{u}) = \frac{1}{2}\|\mathbf{u}\|^2$ is a typical function satisfying the constraints, since $\frac{1}{2}\|\mathbf{u}\|^2 = \frac{1}{2}\|-\mathbf{u}\|^2$, and $\nabla G(\mathbf{u}) = [u_1, \ldots, u_d]^T$ where $g_i(u) = u$, $\forall i \in 1, \ldots, d$. It is also worth mentioning that by choosing $G(\mathbf{u}) = \frac{1}{2}\|\mathbf{u}\|^2$, all updates U_1-U_6 become equivalent:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t \left(\mathrm{E}_{\boldsymbol{\theta}_t} - \hat{\mathrm{E}} \right)$$

which recovers the GD for the log-likelihood function.

When a twice-differentiable function G satisfies the constraints 1 and 2, we have

$$\nabla G(\mathbf{u}) = -\nabla G(-\mathbf{u}) \tag{5.4}$$

$$\nabla G(\mathbf{0}) = \mathbf{0} \tag{5.5}$$

$$\nabla^2 G(\mathbf{u}) = \nabla^2 G(-\mathbf{u}) \tag{5.6}$$

where Eq. 5.6 holds since $\nabla^2 G$ is a diagonal matrix. Given these conditions, we see immediately that U_1 is equivalent to U_3, and U_2 is equivalent to U_5. This way, the types of updates are eventually narrowed down to U_1 and U_2.

To see the relationship between U_1 and U_2, note that the Taylor expansion of $\nabla G(\mathbf{0})$ at

point $\tilde{E} - E_{\theta_t}$ is given by

$$\nabla G(\mathbf{0}) = \nabla G(\tilde{E} - E_{\theta_t}) + \nabla^2 G(\tilde{E} - E_{\theta_t})(E_{\theta_t} - \tilde{E}) + O(\|E_{\theta_t} - \tilde{E}\|^2)$$

Therefore U_1 and U_2 can be regarded as approximations to each other.

Since stochastic optimization is preferred for large-scale training, we replace $\tilde{E}$ and E_{θ_t} with its stochastic estimation $\tilde{E}_t$ and $E_{\theta_t,t}$, where $\tilde{E}_t = \Phi(x_t, y_t)$ and $E_{\theta_t,t} = E_{p_{\theta_t}(Y|x_t)}[\Phi(x_t, Y)]$. Assuming G satisfies the constraints, we re-write U_1 and U_2 as

$$\theta_{t+1} = \theta_t - \lambda_t \nabla^2 G(E_{\theta_t,t} - \tilde{E}_t)(E_{\theta_t} - \tilde{E}_t) \qquad (U_1^*)$$

$$\theta_{t+1} = \theta_t - \lambda_t \nabla G(E_{\theta_t,t} - \tilde{E}_t) \qquad (U_2^*)$$

which will be the focus for the rest of the paper.

5.2.4 Choosing the Convex Function G

We now proceed to choose the actual functional forms of G aiming to make the training procedure more efficient.

A naïve approach would be to choose G from a parameterized convex function family (say the vector p-norm, where $p \geq 1$ is the hyper-parameter), and tune the hyper-parameter on a held-out set hoping to find a proper value that works best for the task at hand. However

this approach is very inefficient, and we would like to choose G in a more principled way.

Although we derived U_1^* and U_2^* from NGD, they can be treated as SGD of two surrogate loss functions $S_1(\boldsymbol{\theta})$ and $S_2(\boldsymbol{\theta})$ respectively (we do not even need to know what the actual functional forms of S_1, S_2 are), whose gradients $\nabla_{\boldsymbol{\theta}} S_1 = \nabla^2 G(\mathrm{E}_{\boldsymbol{\theta}} - \hat{\mathrm{E}})(\mathrm{E}_{\boldsymbol{\theta}} - \hat{\mathrm{E}})$ and $\nabla_{\boldsymbol{\theta}} S_2 = \nabla G(\mathrm{E}_{\boldsymbol{\theta}} - \hat{\mathrm{E}})$ are transformations of the gradient of the log-likelihood (Eq. 5.1). Since the performance of SGD is sensitive to the condition number of the objective function 72, one heuristic for the selection of G is to make the condition numbers of S_1 and S_2 smaller than that of the log-likelihood. However, this is hard to analyze since the condition number is in general difficult to compute. Alternatively, we may select a G so that the second-order information of the log-likelihood can be (approximately) incorporated, as second-order stochastic optimization methods usually converge faster and are insensitive to the condition number of the objective function 72. This is the guideline we follow in this section.

The first convex function we consider is as follows:

$$G_1(\mathbf{u}) = \sum_{i=1}^{d} \frac{u_i}{\sqrt{\epsilon}} \arctan\left(\frac{u_i}{\sqrt{\epsilon}}\right) - \frac{1}{2} \log\left(1 + \frac{u_i^2}{\epsilon}\right)$$

where $\epsilon > 0$ is a small constant free to choose. It can be easily checked that G_1 satisfies

the constraints imposed in Section 5.2.3. The gradients of G_1 are given by

$$\nabla G_1(\mathbf{u}) = \frac{1}{\sqrt{=}} \arctan\left(\frac{u_1}{\sqrt{=}}\right), \ldots, \arctan\left(\frac{u_d}{\sqrt{=}}\right)^T$$

$$\nabla^2 G_1(\mathbf{u}) = \text{diag}\left(\frac{1}{u_1^2 + }, \ldots, \frac{1}{u_d^2 + }\right)$$

In this case, the U_1^* update has the following form (named $U_1^*.G_1$):

$$\boldsymbol{\theta}^i_{t+1} = \boldsymbol{\theta}^i_t - \lambda_t \quad \mathrm{E}^i_{\boldsymbol{\theta}_t,t} - \mathrm{E}^i_t \quad ^2 + \quad ^{-1} \quad \mathrm{E}^i_{\boldsymbol{\theta}_t,t} - \mathrm{E}^i_t$$

where $i = 1, \ldots, d$. This update can be treated as a stochastic second-order optimization procedure, as it scales the gradient Eq. 5.1 by the inverse of its variance in each dimension, and it reminds us of the online Newton step (ONS) 73 algorithm, which has a similar update step. However, in contrast to ONS where the full inverse covariance matrix is used, here we only use the diagonal of the covariance matrix to scale the gradient vector. Diagonal matrix approximation is often used in optimization algorithms incorporating second-order information (for example SGN-QN 186, AdaGrad 28 etc.), as it can be computed orders-of-magnitude faster than using the full matrix ,without sacrificing much performance.

The U_2^* update corresponding to the choice of G_1 has the following form (named $U_2^*.G_1$):

$$\boldsymbol{\theta}^i_{t+1} = \boldsymbol{\theta}^i_t - \lambda_t \arctan\left(\frac{\mathrm{E}^i_{\boldsymbol{\theta}_t,t} - \mathrm{E}^i_t}{\sqrt{=}}\right)$$

Note that the constant $1/\sqrt{-}$ in ∇G_1 has been folded into the learning rate λ_t. Although not apparent at first sight, this update is in some sense similar to $U_1^*.G_1$. From $U_1^*.G_1$ we see that in each dimension, gradients $E^i_{\theta_t} - \tilde{E}_t^{\,i}$ with smaller absolute values get boosted more dramatically than those with larger values (as long as $|E^i_{\theta_t} - \tilde{E}_t^{\,i}|$ is not too close to zero). A similar property also holds for $U_2^*.G_1$, since `arctan` is a sigmoid function, and as long as we choose < 1 so that

$$\frac{d}{du}\arctan\frac{1}{\sqrt{-}}u \;|_{u=0} = \frac{1}{\sqrt{-}} > 1$$

the magnitude of $E^i_{\theta_t} - \tilde{E}_t^{\,i}$ with small absolute value will also get boosted. This is illustrated in Figure 5.1 by comparing functions $\frac{u}{u^2+}$ (where we select $= 0.1$) and $\arctan\frac{1}{\sqrt{-}}u$. Note that for many NLP tasks modeled by CRF, only indicator features are defined. Therefore the value of $E^i_{\theta_t} - \tilde{E}_t^{\,i}$ is bounded by -1 and 1, and we only care about function values on this interval.

Since `arctan` belongs to the sigmoid function family, it is not the only candidate whose corresponding update mimics the behavior of $U_1^*.G_1$, while G still satisfies the constraints given in Section 5.2.3. We therefore consider two more convex functions G_2 and G_3, whose gradients are the `erf` and Gudermannian functions (`gd`) from the sigmoid fam-

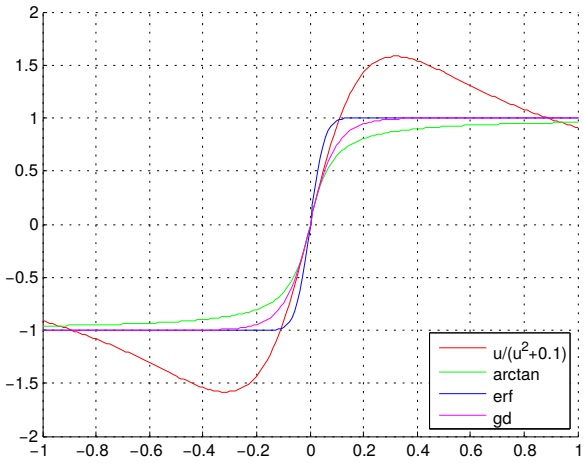


Figure 5.1: Comparison of functions $u/(u^2 + 0.1)$, `arctan`, `erf`, `gd`. The sigmoid functions `arctan`, `erf` and `gd` are normalized so that they have the same gradient as $u/(u^2 + 0.1)$ at $u = 0$ and their values are bounded by -1, 1.

ily respectively:

$$\nabla G_2(\alpha \mathbf{u})_i = \frac{2}{\pi} \int_0^{\alpha u_i} \exp\{-x^2\}\, dx$$

$$\nabla G_3(\beta \mathbf{u})_i = 2 \arctan(\exp\{\beta u_i\}) - \frac{\pi}{2}$$

where $\alpha, \beta \in \mathbb{R}$ are hyper-parameters. In this case, we do not even know the functional form of the functions G_2, G_3, however it can be checked that both of them satisfy the constraints. The reason why we select these two functions from the sigmoid family is that when the gradients of `erf` and `d` are the same as that of `arctan` at point zero, both of them stay on top of `arctan`. Therefore, `erf` and `gd` are able to give stronger boosts to small gradients. This is also illustrated in Figure 5.1.

Applying ∇G_2 and ∇G_3 to U_2^*, we get two updates named $U_2^*.G_2$ and $U_2^*.G_3$ respectively. We do not consider further applying $\nabla^2 G_2$ and $\nabla^3 G_3$ to U_1^*, as the meaning of the

updates becomes less clear.

5.2.5 Adding Regularization

So far we have not considered adding regularization term to the loss functions yet, which is crucial for better generalization. A common choice of the regularization term is the 2-norm (Euclidean metric) of the parameter vector $\frac{C}{2}\|\boldsymbol{\theta}\|^2$, where C is a hyper-parameter specifying the strength of regularization. In our case, however, we derived our algorithms by applying NGD to the Bregman loss functions, which assumes the underlying parameter space to be non-Euclidean. Therefore, the regularization term we add has the form $\frac{C}{2}\boldsymbol{\theta}^T M_{\boldsymbol{\theta}}\boldsymbol{\theta}$, and the regularized loss to be minimized is

$$\frac{C}{2}\boldsymbol{\theta}^T M_{\boldsymbol{\theta}}\boldsymbol{\theta} + B_i \tag{5.7}$$

where B_i, $i = \{1, 2, 3, 4\}$ are the loss functions B_1-B_4.

However, the Riemannian metric $M_{\boldsymbol{\theta}}$ itself is a function of $\boldsymbol{\theta}$, and the gradient of the objective at time t is difficult to compute. Instead, we use an approximation by keeping the Riemannian metric fixed at each time stamp, and the gradient is given by $M_{\boldsymbol{\theta}_t}\boldsymbol{\theta}_t + \nabla_{\boldsymbol{\theta}_t} B_i$. Now if we apply NGD to this objective, the resulting updates will be no different than the SGD for L2-regularized surrogate loss functions S_1, S_2:

$$\boldsymbol{\theta}_{t+1} = (1 - C\lambda_t)\ \boldsymbol{\theta}_t - \frac{\lambda_t}{1 - C\lambda_t}\nabla_{\boldsymbol{\theta}} S_{i,t}(\boldsymbol{\theta}_t)$$

where $i = \{1, 2\}$. It is well-known that SGD for L2-regularized loss functions has an equivalent but more efficient sparse update [76, 187]:

$$\bar{\boldsymbol{\theta}}_t = \frac{\boldsymbol{\theta}_t}{z_t}$$

$$\bar{\boldsymbol{\theta}}_{t+1} = \bar{\boldsymbol{\theta}}_t - \frac{\lambda_t}{(1 - \lambda_t)z_t} \nabla_{\boldsymbol{\theta}} S_{i,t}(\boldsymbol{\theta}_t)$$

$$z_{t+1} = (1 - C\lambda_t) z_t$$

where z_t is a scaling factor and $z_0 = 1$. We then modify U_1^* and U_2^* accordingly, simply by changing the step-size and maintaining a scaling factor.

As for the choice of learning rate λ_t, we follow the recommendation given by [187] and set $\lambda_t = \left[\hat{\lambda}(1 + \hat{\lambda}Ct)\right]^{-1}$, where $\hat{\lambda}$ is calibrated from a small subset of the training data before the training starts.

The final versions of the algorithms developed so far are summarized in Figure 5.2.

5.2.6 Computational Complexity

The proposed algorithms simply transforms the gradient of the log-likelihood, and each transformation function can be computed in constant time, therefore all of them have the same time complexity as SGD ($O(d)$ per update). In cases where only indicator features are defined, the training process can be accelerated by pre-computing the values of $\nabla^2 G$ or ∇G for variables within range $[-1, 1]$ and keep them in a table. During the actual training, the transformed gradient values can be found simply by looking up the table after rounding-

Initialization:

Choose hyper-parameters C, $/\alpha/\beta$ (depending on which convex function to use: G_1, G_2, G_3).

Set $z_0 = 1$, and calibrate $\hat{\lambda}$ on a small training subset.

Algorithm:

for $e = 1$. num_epoch **do**

for $t = 1 \ldots$ **do**

Receive training sample (x_t, y_t)

$\lambda_t = \max(\hat{\lambda}(1 + \hat{\lambda}Ct)^{-1}, \theta_t = z_t\bar{\theta}_t$

Depending on the update strategy selected, update the parameters:

$\bar{\theta}_{t+1} = \bar{\theta}_t - \frac{\lambda_t}{(1-\lambda_t)z_t}\nabla_\theta S_t(\theta_t)$

where $\nabla_\theta S_t(\theta_t)_i$, $i = 1, \ldots,$ is given by

$$E^i_{\theta_t,t} - \bar{E}^i_t + \frac{1}{2}\left(E^i_{\theta_t,t} - \bar{E}^i_t\right)^2 \qquad (U_1^*.G_1)$$

$$\arctan\left(\frac{E^i_{\theta_t,t} - \bar{E}^i_t}{\sqrt{2}}\right) \qquad (U_2^*.G_2)$$

$$\text{erf}\ \alpha(E^i_{\theta_t,t} - \bar{E}^i_t) \qquad (U_2^*.G_2)$$

$$\text{gd}\ \beta(E^i_{\theta_t,t} - \bar{E}^i_t) \qquad (U_2^*.G_3)$$

$z_{t+1} = (1 - C\lambda_t)z_t$

if no more improvement on training set **then**

exit

end if

end for

end for

Figure 5.2: Summary of the proposed algorithms.

off to the nearest entry. This way we do not even need to compute the function values on the fly, which significantly reduces the computational cost.

5.3 Experiments

5.3.1 Settings

We conduct our experiments based on the following settings:

Implementation: We implemented our algorithms based on the CRF suite toolkit 188, in which SGD for the log-likelihood loss with L2 regularization is already implemented. This can be easily done since our algorithm only requires to modify the original gradients. Other parts of the code remain unchanged.

Task: Our experiments are conducted for the widely used CoNLL 2000 chunking shared task 189. The training and test data contain 8939 and 2012 sentences respectively. For fair comparison, we ran our experiments with two standard linear-chain CRF feature sets implemented in the CRF suite. The smaller feature set contains 452,755 features, whereas the larger one contains 7,385,312 features.

Baseline algorithms: We compare the performance of our algorithms summarized in Figure 5.2 with SGD, L-BFGS, and the Passive-Aggressive (PA) algorithm. Except for L-BFGS which is a second-order batch-learning algorithm, we randomly shuffled the data and repeated experiments five times.

Hyper-parameter selection: For convex function G_1 we choose $= 0.1$, correspondingly the `arctan` function in $U_2^*.G_1$ is $\arctan(3.16u)$. We have also experimented the function $\arctan(10u)$ for this update, following the heuristic that the transformation function $\frac{u}{u^2+0.1}$ given by $\nabla^2 G_1$ and $\arctan(10u)$ have consistent gradients at zero, since the U_2^*

update imitates the behavior of U_1^* (the two choices of the `arctan` functions are denoted by `arctan.1` and `arctan.2` respectively). Following the same heuristic, we choose $\alpha = \sqrt[5]{\pi} \approx 8.86$ for the `erf` function and $\beta = 10$ for the `gd` function.

5.3.2 Results

Comparison with the baseline: We compare the performance of the proposed and baseline algorithms on the training and test sets in Figure 5.3 and Figure 5.4, corresponding to small and large feature sets respectively. The plots show the F-scores on the training and test sets for the first 50 epochs. To keep the plots neat, we only show the average F-scores of repeated experiments after each epoch, and omit the standard deviation error bars. For $U_2^*.G_1$ update, only `arctan.1` function is reported here. From the figure we observe that:

1. The strongest baseline is given by SGD. By comparison, PA appears to overfit the training data, while L-BFGS converges very slowly, although eventually it catches up.

2. Although the SGD baseline is already very strong (especially with the large feature set), both the proposed algorithm $U_1^*.G_1$ and $U_2^*.G_1$ outperform SGD and stay on top of the SGD curves most of the time. On the other hand, the $U_2^*.G_1$ update appears to be a little more advantageous than $U_1^*.G_1$.

Comparison of the sigmoid functions: Since `arctan`, `erf` and `gd` are all sigmoid functions and it is interesting to see how their behaviors differ, we compare the updates $U_2^*.G1$ (for both `arctan.1` and `arctan.2`) $U_2^*.G_2$ and $U_2^*.G_3$ in Figure 5.5 and Figure 5.6. The

strongest baseline, SGD, is also included for comparison. From the figures we have the following observations:

1. As expected, the sigmoid functions demonstrated similar behaviors, and performances of their corresponding updates are almost indistinguishable. $U_2^*.G_2$. Similar to $U_2^*.G1$, $U_2^*.G2$ and $U_2^*.G3$ both outperformed the SGD baseline.

2. Performances of $U_2^*.G1$ given by `arctan` functions are insensitive to the choice of the hyper-parameters. Although we did not run similar experiments for `erf` and `gd` functions, similar properties can be expected from their corresponding updates.

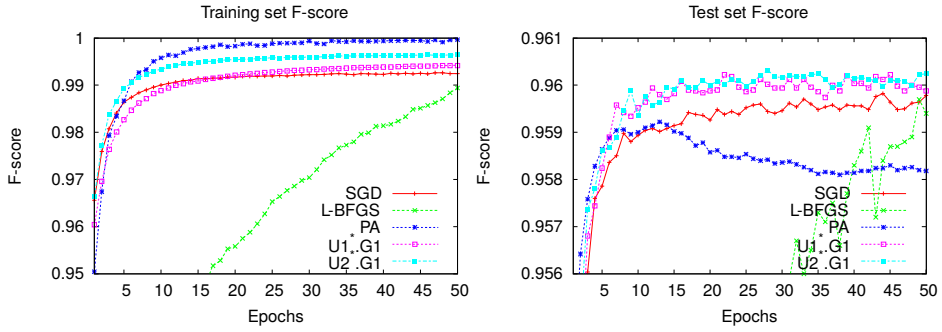


Figure 5.3: F-scores of training and test sets given by the baseline and proposed algorithms, using the small feature set.

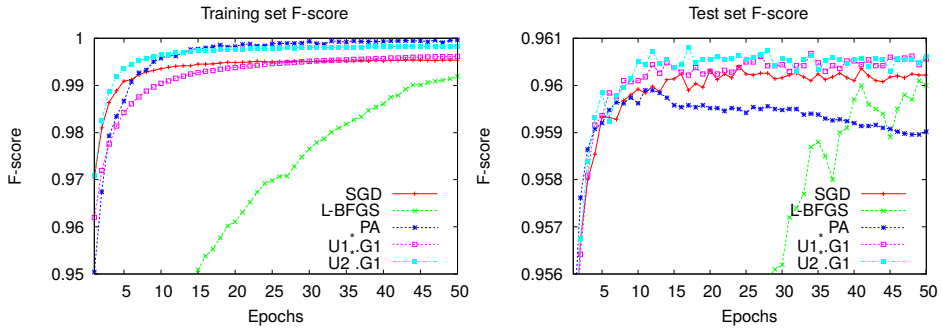


Figure 5.4: F-scores of training and test sets given by the baseline and proposed algorithms, using the large feature set.

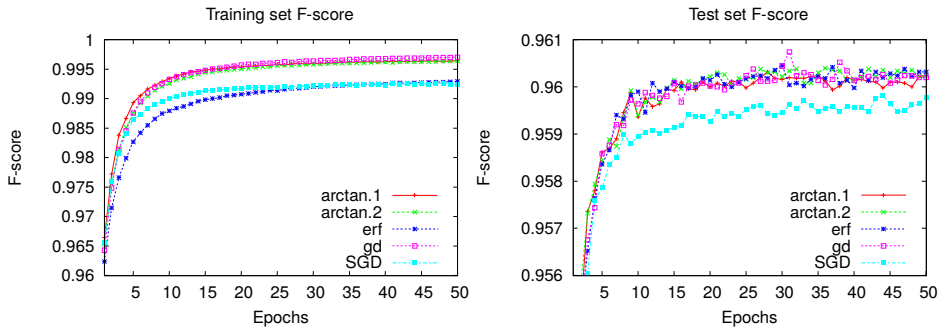


Figure 5.5: F-scores of training and test sets given by $U_2^*.G1$, $U_2^*.G1$ and $U_2^*.G3$, using the small feature set.

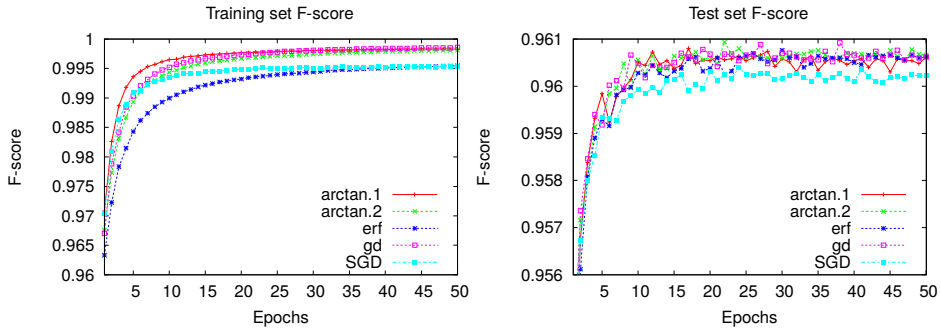


Figure 5.6: F-scores of training and test sets given by $U_2^*.G1$, $U_2^*.G1$ and $U_2^*.G3$, using the large feature set.

Finally, we report in Table 5.2 the F-scores on the test set given by all algorithms after they converge.

Algorithm	F-score % (small)	F-score % (large)
SGD	95.98±0.02	96.02±0.01
PA	95.82±0.04	95.90±0.03
L-BFGS	96.00	96.01
$U_1^*.G_1$	95.99±0.02	96.06±0.03
$U_2^*.G_1$	96.02±0.02	96.06±0.02
$U_2^*.G_1^0$	96.03 ± 0.01	96.06±0.03
$U_2^*.G_2$	96.03±0.02	96.06±0.02
$U_2^*.G_3$	96.02±0.02	96.06±0.02

Table 5.2: F-scores on the test set after algorithm converges, using the small and large feature sets. $U_2^*.G_1$ is the update given by `arctan.1`, and $U_2^*.G_2$ given by `arctan.2`.

5.4 Summary

In this chapter we proposed a novel parameter estimation framework for CRF. By defining loss functions using the Bregman divergences, we are given the opportunity to select convex functions that transform the gradient of the log-likelihood loss, which leads to more effective parameter learning if the function is properly chosen. Minimization of the Bregman loss function is made possible by NGD, thanks to the structure of exponential families. We developed several parameter update strategies which approximately incorporates the second-order information of the log-likelihood, and outperformed baseline algorithms that are already very strong on a popular text chunking task.

Chapter 6

Online Learning in Tensor Space

6.1 Introduction

Many NLP applications use models that try to incorporate a large number of linguistic features so that as much human knowledge of language can be brought to bear on the (prediction) task as possible. This also makes training the model parameters a challenging problem, since the amount of labeled training data is usually small compared to the size of feature sets: the feature weights cannot be estimated reliably.

Most traditional models are linear models, in the sense that both the features of the data and model parameters are represented as vectors in a vector space. Many learning algorithms applied to NLP problems, such as the Perceptron, MIRA, PRO, Rampion etc. that we mentioned in Section 1.3, are based on vector-space models. Such models require learning individual feature weights directly, so that the number of parameters to be estimated is

identical to the size of the feature set. When millions of features are used but the amount of labeled data is limited, it can be difficult to precisely estimate each feature weight.

In this chapter, we shift the model from vector-space to tensor-space. Data can be represented in a compact and structured way using tensors as containers. Tensor representations have been applied to computer vision problems 190, 191 and information retrieval 192 a long time ago. More recently, it has also been applied to parsing 193, 194 and semantic analysis 195. A linear tensor model represents both features and weights in tensor-space, hence the weight tensor can be factorized and approximated by a linear sum of rank-1 tensors. This low-rank approximation imposes structural constraints on the feature weights and can be regarded as a form of regularization. With this representation, we no longer need to estimate individual feature weights directly but only a small number of "bases" instead. This property makes the the tensor model very effective when training a large number of feature weights in a low-resource environment. On the other hand, tensor models have many more degrees of "design freedom" than vector space models. While this makes them very flexible, it also creates much difficulty in designing an optimal tensor structure for a given training set.

We give detailed description of the tensor space model in Section 6.2. Several issues that come with the tensor model construction are addressed in Section 6.3. A tensor weight learning algorithm is then proposed in 6.4. Finally we give our experimental results on a parsing task and analysis in Section 6.5.

6.2 Tensor Space Representation

Most of the learning algorithms for NLP problems are based on vector space models, which represent data as vectors $\boldsymbol{\varphi} \in \mathbb{R}^n$, and try to learn feature weight vectors $\boldsymbol{\theta} \in \mathbb{R}^n$ such that a linear model $y = \boldsymbol{\theta} \cdot \boldsymbol{\varphi}$ is able to discriminate between, say, good and bad hypotheses. While this is a natural way of representing data, it is not the only choice. Below, we reformulate the model from vector to tensor space.

6.2.1 Tensor Space Model

A tensor is a multidimensional array, and is a generalization of commonly used algebraic objects such as vectors and matrices. [1] Specifically, a vector is a 1^{st} order tensor, a matrix is a 2^{nd} order tensor, and data organized as a rectangular cuboid is a 3^{rd} order tensor etc. In general, a D^{th} order tensor is represented as $T \in \mathbb{R}^{n_1 \times n_2 \times \ldots n_D}$, and an entry in T is denoted by $T_{i_1,i_2,\ldots,i_D}$. Different dimensions of a tensor $1, 2, \ldots, D$ are named modes of the tensor.

Using a D^{th} order tensor as container, we can assign each feature of the task a D-dimensional index in the tensor and represent the data as tensors. Of course, shifting from a vector to a tensor representation entails several additional degrees of freedom, e.g., the

[1] This is in fact an over-simplified definition of tensor that treats a tensor just as a high-dimensional data container. A rigorous definition from the perspective of multilinear algebra is as follows: A tensor T of type (r, s) is an element of the tensor space $T^r_s = \underbrace{V \otimes V \otimes \cdots \otimes V}_{r \text{ times}} \otimes \underbrace{V^* \otimes V^* \otimes \cdots \otimes V^*}_{s \text{ times}}$, where V is a vector space and V^* its dual space. In many practical machine learning applications though, the simplified definition is suitable enough.

order D of the tensor and the sizes $\{n_d\}_{d=1}^{D}$ of the modes, which must be addressed when selecting a tensor model. This will be done in Section 6.3.

6.2.2 Tensor Decomposition

Just as a matrix can be decomposed as a linear combination of several rank-1 matrices via SVD, tensors also admit decompositions[2] into linear combinations of "rank-1" tensors. A D^{th} order tensor $A \in \mathbb{R}^{n_1 \times n_2 \times \dots n_D}$ is rank-1 if it can be written as the outer product of D vectors, i.e.

$$A = \mathbf{a}^1 \otimes \mathbf{a}^2 \otimes, \dots, \otimes \mathbf{a}^D$$

where $\mathbf{a}^i \in \mathbb{R}^{n_d}$, $1 \leq d \leq D$A D^{th} order tensor $T \in \mathbb{R}^{n_1 \times n_2 \times \dots n_D}$ can be factorized into a sum of component rank-1 tensors as

$$T = \sum_{r=1}^{R} A_r = \sum_{r=1}^{R} \mathbf{a}_r^1 \otimes \mathbf{a}_r^2 \otimes, \dots, \otimes \mathbf{a}_r^D$$

where R, called the rank of the tensor, is the minimum number of rank-1 tensors whose sum equals T. Via decomposition, one may approximate a tensor by the sum of H major rank-1 tensors with $H \leq R$.

[2]The form of tensor decomposition defined here is named as CANDECOMP/PARAFAC(CP) decomposition 196. Another popular form of tensor decomposition is called Tucker decomposition, which decomposes a tensor into a core tensor multiplied by a matrix along each mode. We focus only on the CP decomposition in this paper.

6.2.3 Linear Tensor Model

In tensor space, a linear model may be written (ignoring a bias term) as

$$f(\boldsymbol{\Theta}) = \boldsymbol{\Theta} \circ \boldsymbol{\Phi},$$

where $\boldsymbol{\Phi} \in \mathsf{R}^{n_1 \times n_2 \times \ldots n_D}$ is the feature tensor, $\boldsymbol{\Theta}$ is the corresponding weight tensor, and $\circ$ denotes the Hadamard product. If $\boldsymbol{\Theta}$ is further decomposed as the sum of H major component rank-1 tensors, i.e. $\boldsymbol{\Theta} \approx \sum_{h=1}^{H} \boldsymbol{\theta}_h^1 \otimes \boldsymbol{\theta}_h^2 \otimes, \ldots, \otimes \boldsymbol{\theta}_h^D$, then

$$\begin{aligned} & f(\boldsymbol{\theta}_1^1, \ldots, \boldsymbol{\theta}_1^D, \ldots, \boldsymbol{\theta}_h^1, \ldots, \boldsymbol{\theta}_h^D) \\ &= \sum_{h=1}^{H} \boldsymbol{\Phi} \times_1 \boldsymbol{\theta}_h^1 \times_2 \boldsymbol{\theta}_h^2 \ldots \times_D \boldsymbol{\theta}_h^D, \end{aligned} \tag{6.1}$$

where $\times_l$ is the l-mode product operator between a D^{th} order tensor T and a vector $\mathbf{a}$ of dimension n_d, yielding a $(D-1)^{\text{th}}$ order tensor such that

$$\begin{aligned} & (T \times_l \mathbf{a})_{i_1,\ldots,i_{l-1},i_{l+1},\ldots,i_D} \\ &= \sum_{i_l=1}^{n_d} T_{i_1,\ldots,i_{l-1},i_l,i_{l+1},\ldots,i_D} \cdot a_{i_l}. \end{aligned}$$

The linear tensor model is illustrated in Figure 6.1.

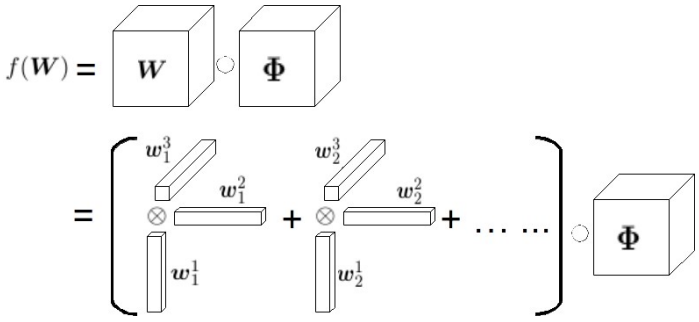


Figure 6.1: A 3^{rd} order linear tensor model. The feature weight tensor Θ can be decomposed as the sum of a sequence of rank-1 component tensors.

6.2.4 Why Learning in Tensor Space?

So what is the advantage of learning with a tensor model instead of a vector model? Consider the case where we have defined 1,000,000 features for our task. A vector space linear model requires estimating 1,000,000 free parameters. However if we use a 2^{nd} order tensor model, organize the features into a 1000×1000 matrix Φ, and use just one rank-1 matrix to approximate the weight tensor, then the linear model becomes

$$f(\theta_1, \theta_2) = \theta_1^T \Phi \theta_2,$$

where $\theta_1, \theta_2 \in R^{1000}$. That is to say, now we only need to estimate 2000 parameters!

In general, if V features are defined for a learning problem, and we (i) organize the feature set as a tensor $\Phi \in R^{n_1 \times n_2 \times \ldots n_D}$ and (ii) use H component rank-1 tensors to approximate the corresponding target weight tensor. Then the total number of parameters to be learned for this tensor model is $H \sum_{d=1}^{D} n_d$, which is usually much smaller than $V = \prod_{d=1}^{D} n_d$ for a traditional vector space model. Therefore we expect the tensor model

to be more effective in a low-resource training environment.

Specifically, a vector space model assumes each feature weight to be a "free" parameter, and estimating them reliably could therefore be hard when training data are not sufficient or the feature set is huge. By contrast, a linear tensor model only needs to learn $H\sum_{d=1}^{D} n_d$ "bases" of the m feature weights instead of individual weights directly. The weight corresponding to the feature $\Phi_{i_1,i_2,\ldots,i_D}$ in the tensor model is expressed as

$$w_{i_1,i_2,\ldots,i_D} = \sum_{h=1}^{H} w^1_{h,i_1} w^2_{h,i_2} \ldots w^D_{h,i_D}, \tag{6.2}$$

where w^j_{h,i_j} is the i_j^{th} element in the vector $\boldsymbol{\theta}^j_h$.

In other words, a true feature weight is now approximated by a set of bases. This reminds us of the well-known low-rank matrix approximation of images via SVD, and we are applying similar techniques to approximate target feature weights, which is made possible only after we shift from vector to tensor space models.

This approximation can be treated as a form of model regularization, since the weight tensor is represented in a constrained form and made highly structured via the rank-1 tensor approximation. Of course, as we reduce the model complexity, e.g. by choosing a smaller and smaller H, the model's expressive ability is weakened at the same time. We will elaborate on this point in Section 6.3.1.

6.3 Tensor Model Construction

To apply a tensor model, we first need to convert the feature vector into a tensor $\mathbf{\Phi}$. Once the structure of $\mathbf{\Phi}$ is determined, the structure of $\mathbf{\Theta}$ is fixed as well. As mentioned in Section 6.2.1, a tensor model has many more degrees of "design freedom" than a vector model, which makes the problem of finding a good tensor structure a nontrivial one.

6.3.1 Tensor Order

The order of a tensor affects the model in two ways: the expressiveness of the model and the number of parameters to be estimated. We assume $H = 1$ in the analysis below, noting that one can always add as many rank-1 component tensors as needed to approximate a tensor with arbitrary precision.

Obviously, the 1^{st} order tensor (vector) model is the most expressive, since it is structureless and any arbitrary set of numbers can always be represented exactly as a vector. The 2^{nd} order rank-1 tensor (rank-1 matrix) is less expressive because not every set of numbers can be organized into a rank-1 matrix. In general, a D^{th} order rank-1 tensor is more expressive than a $(D+1)^{th}$ order rank-1 tensor, as a lower-order tensor imposes less structural constraints on the set of numbers it can express. We formally state this fact as follows:

Theorem 4. *A set of real numbers that can be represented by a $(D+1)^{th}$ order tensor $\mathcal{Q}$ can also be represented by a D^{th} order tensor $\mathcal{P}$, provided $\mathcal{P}$ and $\mathcal{Q}$ have the same volume. But the reverse is not true.*

Proof. For $D = 1$, it is obvious that if a set of real numbers $\{x_1, \ldots, x_n\}$ can be represented by a rank-1 matrix, it can always be represented by a vector, but the reverse is not true.

For $D > 1$, if $\{x_1, \ldots, x_n\}$ can be represented by $P = \mathbf{p}_1 \otimes \mathbf{p}_2 \otimes \ldots \otimes \mathbf{p}_D$, namely $x_i = P_{i_1,\ldots,i_D} = \prod_{d=1}^{D} p_{i_d}^d$, then for any component vector in mode d,

$$[p_1^d, p_2^d, \ldots, p_{n_d^p}^d] = [s_1^d p_1^d, s_2^d p_1^d, \ldots, s_{n_d^p}^d p_1^d]$$

where n_d^p is the size of mode d of P, s_j^d is a constant and $s_j^d = \frac{p_{i_1,\ldots,i_{d-1},j,i_{d+1},\ldots,i_D}}{p_{i_1,\ldots,i_{d-1},1,i_{d+1},\ldots,i_D}}$ Therefore

$$x_i = P_{i_1,\ldots,i_D} = x_{1,\ldots,1} \prod_{d=1}^{D} s_{i_d}^d \tag{6.3}$$

and this representation is unique for a given D (up to the ordering of $\mathbf{p}_j$ and s_j^d in $\mathbf{p}_j$, which simply assigns $\{x_1, \ldots, x_n\}$ with different indices in the tensor), due to the pairwise proportional constraint imposed by x_i/x_j, $i, j = 1, \ldots, n$

If x_i can also be represented by Q, then $x_i = Q_{i_1,\ldots,i_{D+1}} = x_{1,\ldots,1} \prod_{d=1}^{D+1} t_{i_d}^d$, where t_j^d has a similar definition as s_j^d. Then it must be the case that

$$\begin{aligned}
&\exists d_1, d_2 \in \{1, \ldots, D+1\}, d \in \{1, \ldots, D+1\}, d_1 \neq d_2 \neq d, \\
&s.t. \\
&t_{i_{d_1}}^{d_1} t_{i_{d_2}}^{d_2} = s_{i_d}^d, \\
&t_{i_{d_a}}^{d_a} = s_{i_{d_b}}^{d_b}, \quad d_a \neq d_1, d_2, \quad d_b \neq d
\end{aligned} \tag{6.4}$$

since otherwise $\{x_1, \ldots, x_n\}$ would be represented by a different set of factors than those given in Equation (6.3).

Therefore, in order for tensor Q to represent the same set of real numbers that P represents, there needs to exist a vector $[s_1^d, \ldots, s_{n_d}^d]$ that can be represented by a rank-1 matrix as indicated by Eq. (6.4), which is in general not guaranteed.

On the other hand, if $\{x_1, \ldots, x_n\}$ can be represented by Q, namely

$$x_i = Q_{i_1,\ldots,i_{D+1}} = \sum_{d=1}^{D+1} q_{i_d}^d$$

then we can just pick $d_1 \in \{1, \ldots, D\}$, $d_2 = d_1 + 1$ and let

$$\mathbf{q}^0 = [q_1^{d_1} q_1^{d_2}, q_1^{d_1} q_2^{d_2}, \ldots, q_{n_{d_2}}^{d_1} q_{n_{d_1}}^{d_2}]$$

and

$$Q^0 = \mathbf{q}_1 \otimes \ldots \otimes_{d_1-1} \mathbf{q}_{d_1-1} \otimes \mathbf{q}^0 \otimes \mathbf{q}_{d_2+1} \otimes \ldots \otimes_D \mathbf{q}_1$$

Hence $\{x_1, \ldots, x_n\}$ can also be represented by a D^{th} order tensor Q^0. □

On the other hand, tensor order also affects the number of parameters to be trained. Assuming that a D^{th} order has equal size on each mode (we will elaborate on this point in Section 6.3.2) and the volume (number of entries) of the tensor is fixed as V, then the total number of parameters of the model is $DV^{\frac{1}{D}}$. This is a convex function of D, and the

minimum[3] is reached at either $D^* = \lfloor \ln V \rfloor$ or $D^* = \lceil \ln V \rceil$.

Therefore, as D increases from 1 to D^*, we lose more and more of the expressive power of the model but reduce the number of parameters to be trained. However it would be a bad idea to choose a D beyond D^*. The optimal tensor order depends on the nature of the actual problem, and we tune this hyper-parameter on a held-out set.

6.3.2 Mode Size

The size n_d of each tensor mode, $d = 1, \ldots, D$, determines the structure of feature weights a tensor model can precisely represent, as well as the number of parameters to estimate (we also assume $H = 1$ in the analysis below). For example, if the tensor order is 2 and the volume V is 12, then we can either choose $n_1 = 3, n_2 = 4$ or $n_1 = 2, n_2 = 6$. For $n_1 = 3, n_2 = 4$, the numbers that can be precisely represented are divided into 3 groups, each having 4 numbers, that are scaled versions of one another. Similarly for $n_1 = 2, n_2 = 6$, the numbers can be divided into 2 groups with different scales. Obviously, the two possible choices of (n_1, n_2) also lead to different numbers of free parameters (7 vs. 8).

Given D and V, there are many possible combinations of $n_d, d = 1, \ldots, D$, and the optimal combination should indeed be determined by the structure of target features weights. However it is hard to know the structure of target feature weights before learning, and it would be impractical to try every possible combination of mode sizes, therefore we choose

[3] The optimal integer solution can be determined simply by comparing the two function values.

the criterion of determining the mode sizes as minimization of the total number of parameters, namely we solve the problem:

$$\min_{n_1,\ldots,n_D} \sum_{d=1}^{D} n_d \quad s.t \quad \prod_{d=1}^{D} n_d = V$$

The optimal solution is reached when $n_1 = n_2 = \ldots = n_D = V^{\frac{1}{D}}$. Of course it is not guaranteed that $V^{\frac{1}{D}}$ is an integer, therefore we choose $n_d = \lfloor V^{\frac{1}{D}} \rfloor$ or $\lceil V^{\frac{1}{D}} \rceil$, $d = 1, \ldots, D$ such that $\prod_{d=1}^{D} n_d \geq V$ and $\lceil \prod_{d=1}^{D} n_d \rceil - V$ is minimized. The $\lceil \prod_{d=1}^{D} n_d \rceil - V$ extra entries of the tensor correspond to no features and are used just for padding. Since for each n_d there are only two possible values to choose, we can simply enumerate all the possible 2^D (which is usually a small number) combinations of values and pick the one that matches the conditions given above. This way $n_1, \ldots, n_D$ are fully determined.

Here we are only following the principle of minimizing the parameter number. While this strategy might work well with small amount of training data, it is not guaranteed to be the best strategy in all cases, especially when more data is available we might want to increase the number of parameters, making the model more complex so that the data can be more precisely modeled. Ideally the mode size needs to be adaptive to the amount of training data as well as the property of target weights. A theoretically guaranteed optimal approach to determining the mode sizes remains an open problem, and will be explored in our future work.

6.3.3 Number of Rank-1 Tensors

The impact of using $H > 1$ rank-1 tensors is obvious: a larger H increases the model complexity and makes the model more expressive, since we are able to approximate target weight tensor with smaller error. As a trade-off, the number of parameters and training complexity will be increased. To find out the optimal value of H for a given problem, we tune this hyper-parameter too on a held-out set.

6.3.4 Vector to Tensor Mapping

Finally, we need to find a way to map the original feature vector to a tensor, i.e. to associate each feature with an index in the tensor. Assuming the tensor volume V is the same as the number of features, then there are in all $V!$ ways of mapping, which is an intractable number of possibilities even for modest sized feature sets, making it impractical to carry out a brute force search. However while we are doing the mapping, we hope to arrange the features in a way such that the corresponding target weight tensor has approximately a low-rank structure, this way it can be well approximated by very few component rank-1 tensors.

Unfortunately we have no knowledge about the target weights in advance, since that is what we need to learn after all. As a way out, we first run a simple vector-model based learning algorithm (say the Perceptron) on the training data and estimate a weight vector, which serves as a "surrogate" weight vector. We then use this surrogate vector to guide the

design of the mapping. Ideally we hope to find a permutation of the surrogate weights to map to a tensor in such a way that the tensor has a rank as low as possible. However matrix rank minimization is in general a hard problem 197. Therefore, we follow an approximate algorithm given in Figure 6.2a, whose main idea is illustrated via an example in Figure 6.2b.

Basically, what the algorithm does is to divide the surrogate weights into hierarchical groups such that groups on the same level are approximately proportional to each other. Using these groups as units we are able to "fill" the tensor in a hierarchical way. The resulting tensor will have an approximate low-rank structure, provided that the sorted feature weights have roughly group-wise proportional relations.

For comparison, we also experimented a trivial solution which maps each entry of the feature tensor to the tensor just in sequential order, namely φ_0 is mapped to $\Phi_{0,0,\ldots,0}$ φ_1 is mapped to $\Phi_{0,0,\ldots,1}$ etc. This of course ignores correlation between features since the original feature order in the vector could be totally meaningless, and this strategy is not expected to be a good solution for vector to tensor mapping.

6.4 Online Learning Algorithm

We now turn to the problem of learning the feature weight tensor. Here we propose an online learning algorithm similar to MIRA but modified to accommodate tensor models.

Let the model be $f(\boldsymbol{T}) = \boldsymbol{T} \circ \boldsymbol{\Phi}(x, y)$ where $\boldsymbol{T} = \sum_{h=1}^{H} \boldsymbol{\theta}_h^1 \otimes \boldsymbol{\theta}_h^2 \otimes, \ldots, \otimes \boldsymbol{\theta}_h^D$ is the

Input:

Tensor order D, tensor volume V, mode size n_d, $d = 1, \ldots, D$; surrogate weight vector $\boldsymbol{v}$

Let

$\boldsymbol{v}^+ = [v_1^+, \ldots, v_p^+]$ be the non-negative part of $\boldsymbol{v}$

$\boldsymbol{v}^- = [v_1^-, \ldots, v_q^-]$ be the negative part of $\boldsymbol{v}$

Algorithm:

$\tilde{\boldsymbol{v}}^+ = sort(\boldsymbol{v}^+)$ in descending order

$\tilde{\boldsymbol{v}}^- = sort(\boldsymbol{v}^-)$ in ascending order

$$u = V/n_D$$

$$e = p - mod(p, u) \quad f = q - mod(q, u)$$

Construct vector

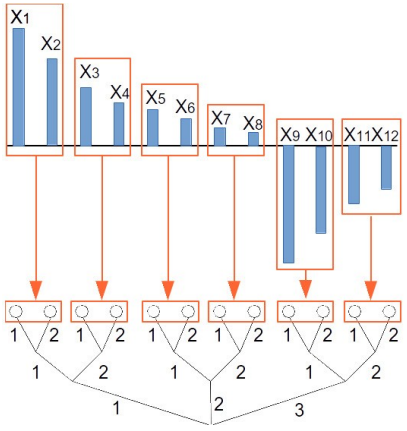
$$\boldsymbol{X} = [\tilde{v}_1^+, \ldots, \tilde{v}_e^+, \tilde{v}_1^-, \ldots, \tilde{v}_f^-, \tilde{v}_{e+1}^+, \ldots, \tilde{v}_p^+, \tilde{v}_{f+1}^-, \ldots, \tilde{v}_q^-]$$

Map X_a, $a = 1, \ldots, p + q$ to the tensor entry $T_{i_1,\ldots,i_D}$, such that

$$a = \sum_{d=1}^{D} (i_d - 1) l_{d-1} + 1$$

where $l_d = l_{d-1} n_d$, and $l_0 = 1$

(a) Mapping a surrogate weight vector to a tensor



(b) Illustration of the algorithm

Figure 6.2: Algorithm for mapping a surrogate weight vector X to a tensor. (6.2a) provides the algorithm; (6.2b) illustrates it by mapping a vector of length $V = 12$ to a $(n_1, n_2, n_3) = (2, 2, 3)$ tensor. The bars X_i represent the surrogate weights — after separately sorting the positive and negative parts — and the labels along a path of the tree correspond to the tensor-index of the weight represented by the leaf resulting from the mapping.

weight tensor, $\mathbf{\Phi}(x, y)$ is the feature tensor for an input-output pair (x, y). Training samples (x_i, y_i), $i = 1, \ldots,$ where x_i is the input and y_i is the reference or oracle hypothesis, are fed to the weight learning algorithm in sequential order. A prediction z_t is made by the model T_t at time t from a set of candidates $Z(x_t)$, and the model updates the weight tensor by solving the following problem:

$$\begin{aligned} &\min_{\boldsymbol{T} \in \mathbb{R}^{n_1 \times n_2 \times \ldots n_D}} \frac{1}{2} \|\boldsymbol{T} - \boldsymbol{T}_t\|^2 + C\xi \\ &s.t. \\ &\mathscr{L}_t \leq \xi, \xi \geq 0 \end{aligned} \tag{6.5}$$

where $\boldsymbol{T}$ is a decomposed weight tensor and

$$\mathscr{L}_t = \boldsymbol{T} \circ \mathbf{\Phi}(x_t, z_t) - \boldsymbol{T} \circ \mathbf{\Phi}(x_t, y_t) + \rho(y_t, z_t)$$

is the structured hinge loss.

This problem setting follows the same "passive-aggressive" strategy as in the original MIRA. To optimize the vectors $\boldsymbol{\theta}_h^d$, $h = 1, \ldots, H$, $d = 1, \ldots, D$, we use a similar iterative strategy as proposed in 198. Basically, the idea is that instead of optimizing $\boldsymbol{\theta}_h^d$ all together, we optimize $\boldsymbol{\theta}_1^1, \boldsymbol{\theta}_1^2, \ldots, \boldsymbol{\theta}_H^D$ in turn. While we are updating one vector, the rest are fixed. For the problem setting given above, each of the sub-problems that need to be solved is convex, and according to 198 the objective function value will decrease after each individual weight update and eventually this procedure will converge.

We now give this procedure in more detail. Denote the weight vector of the d^{th} mode of the h^{th} tensor at time t as $\boldsymbol{w}_{h,t}^d$. We will update the vectors in turn in the following order: $\boldsymbol{w}_{1,t}^1, \ldots, \boldsymbol{w}_{1,t}^D, \boldsymbol{w}_{2,t}^1, \ldots, \boldsymbol{w}_{2,t}^D, \ldots, \boldsymbol{w}_{H,t}^1, \ldots, \boldsymbol{w}_{H,t}^D$. Once a vector has been updated, it is fixed for future updates.

By way of notation, define

$$\boldsymbol{\Theta}_{h,t}^d = \boldsymbol{\theta}_{h,t+1}^1 \otimes, \ldots, \otimes \boldsymbol{\theta}_{h,t+1}^d \otimes \boldsymbol{\theta}_{h,t}^d \otimes, \ldots, \otimes \boldsymbol{\theta}_{h,t}^D$$

and let $\boldsymbol{\Theta}_{h,t}^{D+1} \triangleq \boldsymbol{\theta}_{h,t+1}^1 \otimes, \ldots, \otimes \boldsymbol{\theta}_{h,t+1}^D$.

$$\bar{\boldsymbol{\Theta}}_{h,t}^d = \boldsymbol{\theta}_{h,t+1}^1 \otimes, \ldots, \otimes \boldsymbol{\theta}_{h,t+1}^d \otimes \boldsymbol{\theta}^d \otimes, \ldots, \otimes \boldsymbol{\theta}_{h,t}^D$$

where $\boldsymbol{\theta}^d \in \mathbb{R}^{n_d}$,

$$\begin{aligned} \boldsymbol{T}_{h,t}^d &= \sum_{h'=1}^{h-1} \boldsymbol{\Theta}_{h',t}^{D+1} + \boldsymbol{\Theta}_{h,t}^d + \sum_{h'=h+1}^{H} \boldsymbol{\Theta}_{h',t}^1 \\ \bar{\boldsymbol{T}}_{h,t}^d &= \sum_{h'=1}^{h-1} \boldsymbol{\Theta}_{h',t}^{D+1} + \bar{\boldsymbol{\Theta}}_{h,t}^d + \sum_{h'=h+1}^{H} \boldsymbol{\Theta}_{h',t}^1 \end{aligned} \tag{6.6}$$

$$\boldsymbol{\varphi}_{h,t}^d(x, y) = \boldsymbol{\Phi}(x, y)_2 \times \boldsymbol{\theta}_{h,t+1}^2 \ldots \times_{d-1} \boldsymbol{\theta}_{h,t+1}^{d-1} \times_{d+1} \boldsymbol{\theta}_{h,t}^{d+1} \ldots \times_D \boldsymbol{\theta}_{h,t}^D \tag{6.7}$$

In order to update from $\boldsymbol{w}_{h,t}^d$ to get $\boldsymbol{w}_{h,t+1}^d$, the sub-problem to solve is:

$$\begin{aligned}
&\min_{\boldsymbol{\theta}^d \in \mathbb{R}^{n_d}} \frac{1}{2}\|\hat{\boldsymbol{p}}_{h,t}^d - \boldsymbol{T}_{h,t}^d\|^2 + C\xi \\
&= \min_{\boldsymbol{\theta}^d \in \mathbb{R}^{n_d}} \frac{1}{2}\|\hat{\boldsymbol{\Theta}}_{h,t}^d - \boldsymbol{\Theta}_{h,t}^d\|^2 + C\xi \\
&= \min_{\boldsymbol{w}^d \in \mathbb{R}^{n_d}} \frac{1}{2}\beta_{h,t+1}^1 \cdots \beta_{h,t+1}^{d-1}\beta_{h,t}^{d+1} \cdots \beta_{h,t}^D \|\boldsymbol{w}^d - \boldsymbol{w}_{h,t}^d\|^2 + C\xi \\
s.t. \quad & \mathscr{L}_{h,t}^d \le \xi, \ \xi \ge 0.
\end{aligned}$$

where

$$\begin{aligned}
\beta_{h,t}^d &= \|\boldsymbol{\theta}_{h,t}^d\|^2 \\
\mathscr{L}_{h,t}^d &= \hat{\boldsymbol{p}}_{h,t}^d \circ \boldsymbol{\Phi}(x_t, z_t) - \hat{\boldsymbol{p}}_{h,t}^d \circ \boldsymbol{\Phi}(x_t, y_t) + \rho(y_t, z_t) \\
&= \boldsymbol{\theta}_{h,t}^d \cdot \left(\boldsymbol{\varphi}_{h,t}^d(x_t, z_t) - \boldsymbol{\varphi}_{h,t}^d(x_t, y_t) - \left(\sum_{h'=1}^{h-1} \boldsymbol{\Theta}_{h',t}^{D+1} + \sum_{h'=h+1}^{H} \boldsymbol{\Theta}_{h',t}^1 \right) \circ \right. \\
&\quad \left. (\boldsymbol{\Phi}(x_t, y_t) - \boldsymbol{\Phi}(x_t, z_t)) \right) + \rho(y_t, z_t)
\end{aligned}$$

Letting

$$\Delta\boldsymbol{\varphi}_{h,t}^d \triangleq \boldsymbol{\varphi}_{h,t}^d(x_t, y_t) - \boldsymbol{\varphi}_{h,t}^d(x_t, z_t)$$

and

$$s_{h,t}^d \triangleq \left(\sum_{h'=1}^{h-1} \boldsymbol{\Theta}_{h',t}^{D+1} + \sum_{h'=h+1}^{H} \boldsymbol{\Theta}_{h',t}^1 \right) \circ (\boldsymbol{\Phi}(x_t, y_t) - \boldsymbol{\Phi}(x_t, z_t))$$

we may compactly write

$$\mathscr{L}^d_{h,t} = \rho(y_t, \bar{z}_t) - \$^d_{h,t} - \boldsymbol{\theta}^d \cdot \Delta\boldsymbol{\varphi}^d_{h,t}.$$

This convex optimization problem is just like the original MIRA and may be solved in a similar way. The updating strategy for $\boldsymbol{w}^d_{h,t}$ is derived as

$$\begin{aligned} \boldsymbol{w}^d_{h,t+1} &= \boldsymbol{w}^d_{h,t} + \tau\Delta\boldsymbol{\varphi}^d_{h,t} \\ \tau &= \min\left(C, \frac{\rho(y_t, \bar{z}_t) - \boldsymbol{T}^d_{h,t} \circ (\boldsymbol{\Phi}(x_t, y_t) - \boldsymbol{\Phi}(x_t, \bar{z}_t))}{k\Delta\boldsymbol{\varphi}^d_{h,t}k^2}\right) \end{aligned} \tag{6.8}$$

The initial vectors $\boldsymbol{\theta}^j_{h,1}$ cannot be made all zero, since otherwise the l-mode product in Equation (6.7) would yield all zero $\boldsymbol{\varphi}^d_{h,t}(x, y)$and the model would never get a chance to be updated. Therefore, we initialize the entries of $\boldsymbol{\theta}^j_{h,1}$ uniformly such that the Frobenius-norm of the weight tensor $\boldsymbol{\Theta}$ is unity.

We call the algorithm above "Tensor-MIRA" and abbreviate it as T-MIRA.

6.5 Experiments

In this section we shows empirical results of the training algorithm on a parsing task. We used the Charniak parser 37 for our experiment, and we used the proposed algorithm to train the reranking feature weights. For comparison, we also investigated training the reranker with Perceptron and MIRA.

6.5.1 Experimental Settings

To simulate a low-resource training environment, our training sets were selected from sections 2-9 of the Penn WSJ treebank, section 24 was used as the held-out set and section 23 as the evaluation set. We applied the default settings of the parser. There are around $V = 1.33$ million features in all defined for reranking, and the n-best size for reranking is set to 50. We selected the parse with the highest f-score from the 50-best list as the oracle.

We would like to observe from the experiments how the amount of training data as well as different settings of the tensor degrees of freedom affects the algorithm performance. Therefore we tried all combinations of the following experimental parameters:

Parameters	Settings
Training data (m)	Sec. 2, 2-3, 2-5, 2-9
Tensor order (D)	2, 3, 4
# rank-1 tensors (H)	1, 2, 3
Vec. to tensor mapping	approximate, sequential

Here "approximate" and "sequential" means using, respectively, the algorithm given in Figure 6.2 and the sequential mapping mentioned in Section 6.3.4. According to the strategy given in 6.3.2, once the tensor order and number of features are fixed, the sizes of modes and total number of parameters to estimate are fixed as well, as shown in the tables below:

D	Size of modes	Number of parameters
2	1155×1155	2310
3	$110 \times 110 \times 111$	331
4	$34 \times 34 \times 34 \times 34$	136

6.5.2 Results and Analysis

The F-scores of the held-out and evaluation set given by T-MIRA as well as the Perceptron and MIRA baseline are given in Table 6.1. From the results, we have the following observations:

1. When very few labeled data are available for training (compared with the number of features), T-MIRA performs much better than the vector-based models MIRA and Perceptron. However as the amount of training data increases, the advantage of T-MIRA fades away, and vector-based models catch up. This is because the weight tensors learned by T-MIRA are highly structured, which significantly reduces model/training complexity and makes the learning process very effective in a low-resource environment, but as the amount of data increases, the more complex and expressive vector-based models adapt to the data better, whereas further improvements from the tensor model is impeded by its structural constraints, making it insensitive to the increase of training data.

2. To further contrast the behavior of T-MIRA, MIRA and Perceptron, we plot the F-scores on both the training and held-out sets given by these algorithms after each training epoch in Figure 6.3. The plots are for the experimental setting with mapping=surrogate, # rank-1 tensors=2, tensor order=2, training data=sections 2-3. It is clearly seen that both MIRA and Perceptron do much better than T-MIRA on the training set. Nevertheless, with a huge number of parameters to fit a limited amount

of data, they tend to over-fit and give much worse results on the held-out set than T-MIRA does.

As an aside, observe that MIRA consistently outperformed Perceptron, as expected.

3. Properties of linear tensor model: The heuristic vector-to-tensor mapping strategy given by Figure 6.2 gives consistently better results than the sequential mapping strategy, as expected.

   To make further comparison of the two strategies, in Figure 6.4 we plot the 20 largest singular values of the matrices which the surrogate weights (given by the Perceptron after running for 1 epoch) are mapped to by both strategies (from the experiment with training data sections 2-5). From the contrast between the largest and the 2^{nd}-largest singular values, it can be seen that the matrix generated by the first strategy approximates a low-rank structure much better than the second strategy. Therefore, the performance of T-MIRA is influenced significantly by the way features are mapped to the tensor. If the corresponding target weight tensor has internal structure that makes it approximately low-rank, the learning procedure becomes more effective.

   The best results are consistently given by 2^{nd} order tensor models, and the differences between the 3^{rd} and 4^{th} order tensors are not significant. As discussed in Section 6.3.1, although 3^{rd} and 4^{th} order tensors have less parameters, the benefit of reduced training complexity does not compensate for the loss of expressiveness. A 2^{nd} order tensor has already reduced the number of parameters from the original 1.33 million

to only 2310, and it does not help to further reduce the number of parameters using higher order tensors.

4. As the amount of training data increases, there is a trend that the best results come from models with more rank-1 component tensors. Adding more rank-1 tensors increases the model's complexity and ability of expression, making the model more adaptive to larger data sets.

Mapping	Approximate									Sequential								
Rank-1 tensors	1			2			3			1			2			3		
Tensor order	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4
Held-out score	**89.43**	89.16	89.22	89.16	89.21	89.24	89.27	89.14	89.24	**89.21**	88.90	88.89	89.13	88.88	88.88	89.15	88.87	88.99
Evaluation score	**89.83**									89.69								
MIRA	88.57																	
Percep	88.23																	

(a) Training data: Section 2 only

Mapping	Approximate									Sequential								
Rank-1 tensors	1			2			3			1			2			3		
Tensor order	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4
Held-out score	89.26	89.06	89.12	**89.33**	89.11	89.19	89.18	89.14	89.15	89.2	89.01	88.82	**89.24**	88.94	88.95	89.19	88.91	88.98
Evaluation score	**90.02**									89.82								
MIRA	89.00																	
Percep	88.59																	

(b) Training data: Section 2-3

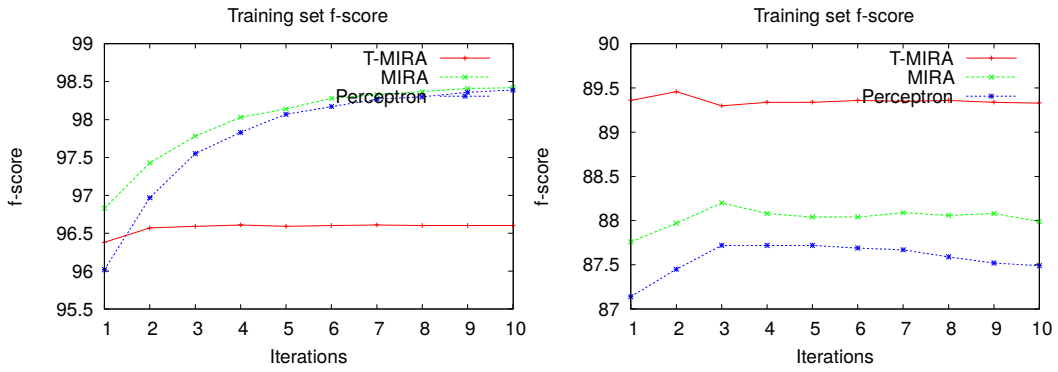
Mapping	Approximate									Sequential								
Rank-1 tensors	1			2			3			1			2			3		
Tensor order	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4
Held-out score	89.40	89.44	89.17	**89.5**	89.37	89.18	89.47	89.32	89.18	89.23	89.03	88.93	**89.24**	88.98	88.94	89.16	89.01	88.85
Evaluation score	**89.96**									89.78								
MIRA	89.49																	
Percep	89.10																	

(c) Training data: Section 2-5

Mapping	Approximate									Sequential								
Rank-1 tensors	2			3			4			2			3			4		
Tensor order	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4
Held-out score	89.43	89.23	89.06	89.37	89.23	89.1	**89.44**	89.22	89.06	89.21	88.92	88.94	89.23	88.94	88.93	**89.23**	88.95	88.93
Evaluation score	**89.95**									89.84								
MIRA	89.95																	
Percep	89.77																	

(d) Training data: Section 2-9

Table 6.1: Parsing F-scores. Tables (a) to (d) correspond to training data with increasing size. The upper-part of each table shows the T-MIRA results with different settings, the lower-part shows the MIRA and Perceptron baselines. The evaluation scores come from the settings indicated by the best held-out scores. The best results on the held-out and evaluation data are marked in bold.



(a) Training set **(b)** Held-out set

Figure 6.3: F-scores given by three algorithms on training and held-out set (see text for the setting).

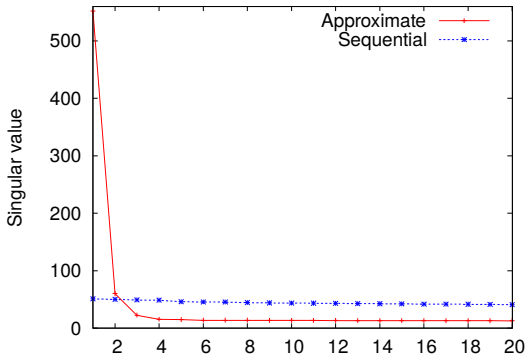


Figure 6.4: The top 20 singular values of the surrogate weight matrices given by two mapping algorithms.

6.6 Summary

In this paper, we reformulated the traditional linear vector-space models as tensor-space models, and proposed an online learning algorithm named Tensor-MIRA. A tensor-space model is a compact representation of data, and via rank-1 tensor approximation, the weight tensor can be made highly structured hence the number of parameters to be trained is significantly reduced. This can be regarded as a form of model regularization.Therefore, compared with the traditional vector-space models, learning in the tensor space is very effective

when a large feature set is defined, but only small amount of training data is available. Our experimental results corroborated this argument.

Chapter 7

Conclusion

7.1 Summary of the Thesis

The contribution of the thesis is divided into three parts: Extensions and detailed study of multiplicative online learning algorithms, efficient training procedure for CRF with novel objectives, low-resource environment online learning in tensor space.

Multiplicative learning algorithms contrasts with the more commonly used additive algorithms in that they favor sparse solutions, and this makes them potentially more suitable candidate algorithms for large-scale discriminative training where millions of discriminative sparse features are defined but only a tiny portion play a significant role, as shown in Chapter 3. The existing general online learning framework OMD is suitable for deriving additive algorithms, but our proposed GMU framework (Chapter 4) is more suitable for deriving multiplicative algorithms. The reason why they yield different update styles

is that they choose different divergences (Bregman vs. f-divergence) as proximity measure. The study of information geometry shows that Bregman and f-divergences induce different Riemannian geometric structures. Inspired by these analysis we are interested in giving analysis and interpretations to general online learning algorithms from geometric perspectives. The flat and curved manifold assumptions of OMD and GMU help us to better understand the nature of the algorithms their different behaviors.

Geometric study not only enables us to gain deeper insights into learning algorithm properties, but also helps to make algorithms more efficient. In Chapter 5 we proposed a novel CRF (or general log-linear models) training objective. The objective is flexible and the training procedure can be made more efficient if proper convex functions are chosen. Direct optimization of the training objective can be a tough problem using conventional GD-based methods, however it is made a lot easier if the geometry of the distribution space is taken into consideration, as can be seen by choosing NGD instead of GD for optimizing.

We also extended online learning from vector to tensor space. Tensor, as a concept studied in multi-linear algebra, also plays a key role in differential geometric analysis (though in a much more sophisticated way than we use it here). For our purpose of efficient learning under low-resource environments, tensor serves as a compact and structured data container. Such a learning paradigm is suitable for cases where millions of features are defined, as the dimensionality reduction via low-rank tensor approximation is dramatic. Learning with such a structure could be difficult if batch algorithms are used, but is made easier in the setting of online learning, as is shown in Chapter 6.

7.2 Future Research Directions

Both online learning and differential geometry are huge and interesting topics. In this thesis we are only able to look into a small part of the two fields and set up a general framework for multiplicative learning. Many further studies and extensions can be made in both directions. For example, similar to the improvements and variations of OMD introduced in Section 2.1.6, GMU may also be extended in several dimensions, for example in cases of composite objectives, designing adaptive learning rates etc. On the other hand, the geometric analysis of GMU can be further refined. The non-flat manifold assumption made by GMU (Section 4.3) makes the analysis difficult and we are not able to give closed-form solutions to the corresponding Riemannian metrics and connections. However, it is possible that some metric and connections with constrained structures may be determined and recovers some specific multiplicative algorithms. This way we will gain a better understanding of the geometry corresponding to GMU and give more precise analysis.

For the CRF training with NGD, proper choice of the convex functions is critical to the performance of the proposed algorithms, and is an interesting problem that is worthy of further investigation. While we selected convex functions with the motivation to reduce the types of updates and incorporate approximate second-order information, there are certainly more possible choices and the performance could be improved via careful theoretical analysis. On the other hand, instead of choosing a convex function a priori, we may rely on some heuristics from the actual data and choose a function tailored for the task at hand.

Finally, one interesting problem for the tensor-space online learning that merits further

investigation is how to determine optimal model sizes. The challenge of applying a tensor model comes from finding a proper tensor structure for a given problem, and the key to solving this problem is to find a balance between the model complexity (indicated by the order and sizes of models) and the number of parameters. Developing a theoretically guaranteed approach of finding the optimal structure for a given task will make the tensor model not only perform well in low-resource environments, but adaptive to larger data sets.

Appendix A

Convex Analysis

Convex analysis is a broad topic and lays the foundation of convex optimization methods. We omit many details of this topic and focus mainly on the convex function properties that are tightly related to the analysis of online learning and information geometry introduced in Chapter 2. General introductions to this topic can be found in many textbooks, for example 199–201, as well as Appendix A in 60.

A.1 Convex Functions

- A set Ω is convex if for two vectors $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \Omega$, the vector $\alpha \boldsymbol{w}_1 + (1 - \alpha)\boldsymbol{w}_2 \in \Omega$, where $\alpha \in [0, 1]$. Given a convex set $\Omega \subseteq \mathrm{R}^d$, a function $f : \Omega \to \mathrm{R}$ is convex if for any $\boldsymbol{x}, \boldsymbol{y} \in \Omega$ and $\alpha \in [0, 1]$

$$f\left((1 - \alpha)\boldsymbol{x} + \alpha \boldsymbol{y}\right) \leq (1 - \alpha) f(\boldsymbol{x}) + \alpha f(\boldsymbol{y})$$

- A vector $\boldsymbol{v}$ is a subgradient of f at $\boldsymbol{x}$ if

$$\forall \boldsymbol{y} \in \Omega,\ f(\boldsymbol{y}) - f(\boldsymbol{x}) \geq \boldsymbol{v}^{T}(\boldsymbol{y} - \boldsymbol{x})$$

The set of $\boldsymbol{v}$ that satisfy this condition, denoted by $\partial f(\boldsymbol{x})$, is the set of all subgradients of f at $\boldsymbol{x}$. When f is differentiable at $\boldsymbol{x}$, then $\partial f(\boldsymbol{x})$ contains a single element, namely the gradient of f at $\boldsymbol{x}$, denoted by $\nabla f(\boldsymbol{x})$.

- A function f over a convex set Ω is called α-strongly convex with respect to a norm $\|\cdot\|$ if for all $\boldsymbol{x}, \boldsymbol{y} \in \Omega$

$$\forall \boldsymbol{v} \in \partial f,\ f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{v}^{T}(\boldsymbol{y} - \boldsymbol{x}) + \frac{\alpha}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^{2}$$

and is β-smooth with respect to the norm $\|\cdot\|$ if

$$\forall \boldsymbol{v} \in \partial f,\ f(\boldsymbol{y}) \leq f(\boldsymbol{x}) + \boldsymbol{v}^{T}(\boldsymbol{y} - \boldsymbol{x}) + \frac{\beta}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^{2}$$

- A function f is called L-Lipschitz over a set S with respect to a norm $\|\cdot\|$ if for all $\boldsymbol{x}, \boldsymbol{y} \in S |f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|$

- Assume that f is differentiable, then α-convexity is equivalent to the following condition:

$$(\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}))^{T}(\boldsymbol{x} - \boldsymbol{y}) \geq \alpha\|\boldsymbol{x} - \boldsymbol{y}\|^{2}$$

Additionally, if f is twice differentiable, then α-convexity is equivalent to the following

condition:

$$\nabla^2 f(\boldsymbol{x}) \quad \frac{\alpha}{2} I_d$$

where I_d is the identity matrix of dimension d, $A \quad B$ means the matrix $A - B$ is positive semi-definite.

- Assume that f is differentiable, then β-smoothness is equivalent to the β-Lipschitz property of the gradient ∇f :

$$k\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})k \leq \beta k\boldsymbol{x} - \boldsymbol{y}k$$

Additionally, if f is twice differentiable, then β-smoothness is equivalent to the following condition:

$$\frac{\beta}{2} I_d \quad \nabla \quad ^2 f(\boldsymbol{x})$$

- When a function is both α-convex and β-smooth, the condition number of f is defined as $\frac{\beta}{\alpha} \geq 1$. Roughly speaking, the condition number characterizes the "easiness" of optimizing f using first-order methods: a smaller condition number (well-conditioned) implies faster convergence using gradient descent, whereas a large condition number (ill-conditioned) requires many iterations.

A.2 Legendre Transformation

- The Legendre transformation of a strictly convex function $f : \Omega \to \mathsf{R}$ is defined as

$$f^*(\boldsymbol{y}) = \sup_{\boldsymbol{x}\in\Omega} \boldsymbol{x}^T\boldsymbol{y} - f(\boldsymbol{x}) \tag{A.1}$$

and f^* (as a result of this operation) is called the convex conjugate or Fenchel conjugate of f. Since f^* is defined as a supremum of linear functions, itself is also convex. If f is closed then $(f^*)^* = f$.

- So what does the Legendre transformation do and why is it given in the form of Eq. A.1? The answer is that the Legendre transformation provides a different way of representing a convex function, in which all the information about the function is losslessly preserved. A graphical illustration is given in Figure A.1.

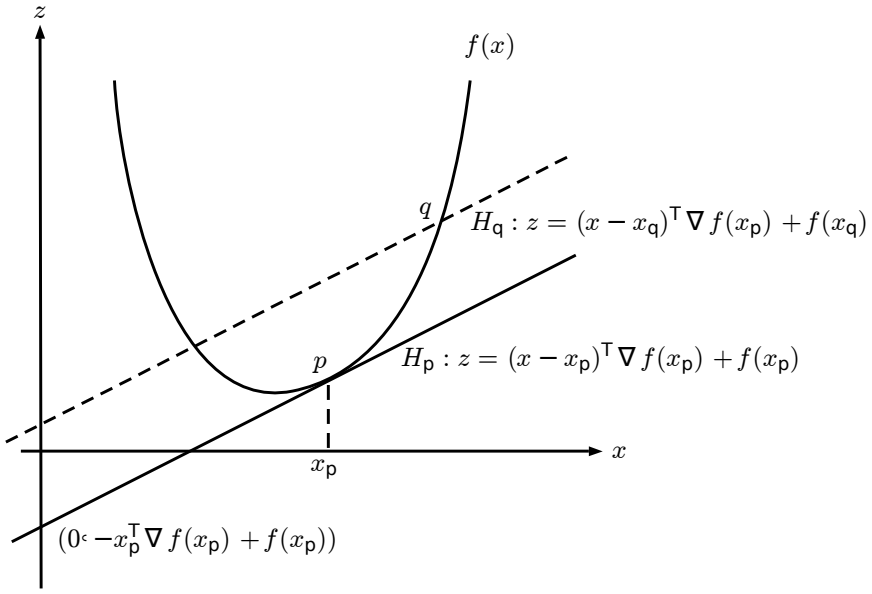


Figure A.1: Illustration of the Legendre transformation. See text for details.

APPENDIX A. CONVEX ANALYSIS

Consider a strictly convex function $f(\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{R}^d$. The most direct way to describe the shape of f is to represent each point p on the contour of the function using its coordinate $(\boldsymbol{x}_p, f(\boldsymbol{x}_p))$. To find a different description of $f(\boldsymbol{x})$, a key observation that there is a one-to-one mapping between $\boldsymbol{x}_p$ and the corresponding gradient $\boldsymbol{y}_p \triangleq \nabla f(\boldsymbol{x}_p)$ due to the convexity of f, which means that the role of $\boldsymbol{x}_p$ can be replaced by $\boldsymbol{y}_p$. However on the other hand, which quantity should be used to replace the role of $f(\boldsymbol{x}_p)$? Consider all parallel hyperplanes H_q with slope equal to $\boldsymbol{y}_p$ and have at least one intersection point q with $f(\boldsymbol{x})$, written as $H_q : z = (\boldsymbol{x} - \boldsymbol{x}_q)^T \nabla f(\boldsymbol{x}_p) + f(\boldsymbol{x}_q)$. Among all these hyperplanes, none of them contains the information of $f(\boldsymbol{x}_p)$ except the one that is tangent with $f(\boldsymbol{x})$ at $\boldsymbol{x}_p$: $H_p : z = (\boldsymbol{x} - \boldsymbol{x}_p)^T \nabla f(\boldsymbol{x}_p) + f(\boldsymbol{x}_p)$. What is more, H_p is the unique hyperplane that minimizes the value of z at the intersection of H_p and the z axis: $z = -\boldsymbol{x}_p^T \nabla f(\boldsymbol{x}_p) + f(\boldsymbol{x}_p)$. Therefore, the point p can be characterized by

$$p = \inf_{q \in \text{contour} f} -\boldsymbol{x}_q^T \boldsymbol{y} + f(\boldsymbol{x}_q) = \sup_{q \in \text{contour} f} \boldsymbol{x}_q^T \boldsymbol{y} - f(\boldsymbol{x}_q)$$

Now we see that given $\boldsymbol{y}_p = \nabla f(\boldsymbol{x}_p)$, we may determine the point p by computing $f^*(\boldsymbol{y}_p) = \sup_{\boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{y}_p - f(\boldsymbol{x})$. In other words, instead of representing points on the contour of f using $(\boldsymbol{x}, f(\boldsymbol{x}))$, they can be equivalently represented by $(\boldsymbol{y}, f^*(\boldsymbol{y}))$.

The Legendre transformation can be viewed as a coordinate system transformation from $\boldsymbol{x}$ to $\boldsymbol{y}$ via $\boldsymbol{y} = \nabla f(\boldsymbol{x})$, which played a critical role in the analysis of the dually flat manifold introduced in Section 2.2.2.2.

- Following the definition of convex conjugate, the following inequality holds:

$$f(\boldsymbol{x}) + f^*(\boldsymbol{y}) - \boldsymbol{x}^T\boldsymbol{y} \geq 0 \tag{A.2}$$

This is called the Fenchel-Young inequality. Note that this inequality can be used to define the Bregman divergence (in its dual form), see Eq. C.1.

The Fenchel-Young inequality holds with equality for $\boldsymbol{z} \in \partial f(\boldsymbol{x})$

$$f(\boldsymbol{x}) + f^*(\boldsymbol{z}) = \boldsymbol{x}^T\boldsymbol{z}$$

- If f is differentiable, the following relation holds:

$$\nabla f^*(\boldsymbol{x}) = (\nabla f)^{-1}(\boldsymbol{x})$$

Additionally, if f is twice differentiable, and let $\boldsymbol{y} = \nabla f(\boldsymbol{x})$, the following relation holds:

$$\nabla^2 f^*(\boldsymbol{y}) = \nabla^2 f^{-1}(\boldsymbol{x}) \tag{A.3}$$

- If f is α-strongly convex, then f^* is $\frac{1}{\alpha}$-smooth; If f is β-smooth, then f^* is $\frac{1}{\beta}$-strongly convex.

- The Legendre transform has the following properties:

- *Scaling*:

$$F(\boldsymbol{x}) = \lambda f(\boldsymbol{x}) \Rightarrow F^*(\boldsymbol{y}) = \lambda f^*\left(\frac{\boldsymbol{y}}{\lambda}\right)$$
$$F(\boldsymbol{x}) = f(\lambda \boldsymbol{x}) \Rightarrow F^*(\boldsymbol{y}) = f^*\left(\frac{\boldsymbol{y}}{\lambda}\right)$$

- *Translation*:

$$F(\boldsymbol{x}) = f(\boldsymbol{x}) + \lambda \Rightarrow F^*(\boldsymbol{y}) = f^*(\boldsymbol{y}) - \lambda$$
$$F(\boldsymbol{x}) = f(\boldsymbol{x} + \boldsymbol{x}_0) \Rightarrow F^*(\boldsymbol{y}) = f^*(\boldsymbol{y}) - \boldsymbol{y}^T \boldsymbol{x}_0$$

- *Inversion (for scalar variables)*:

$$F(x) = f^{-1}(x) \Rightarrow F^*(y) = -yf^*\left(\frac{1}{y}\right)$$

Finally we note that in the context of information geometry, a good introduction to Legendre transformation can be found in 202, from which much content of this section is adopted.

Appendix B

Fundamentals of Riemannian Geometry

Riemannian geometry, originated with the insight of Bernhard Riemann in the 19^{th} century, is a branch of differential geometry which studies Riemannian manifolds (manifolds equipped with Riemannian metrics). After more than a century's development, Riemannian geometry has become a huge and mature subject. It is impossible to cover many aspects of this geometry in this appendix, and we briefly introduce some basic concepts related to information geometry (Section 2.2) with lots of details omitted. General introductions to this interesting topic are widely available, for example 203–205, as well as Chapter 1 in 116.

B.1 Smooth Manifolds

A manifold M of dimension d (or d-manifold), is a topological space with the following properties:

1. M is Hausdorff.[1]

2. M is locally Euclidean of dimension d.[2]

3. M has a countable basis of open sets.

Since M is locally Euclidean, we may define a set A of coordinate systems, each of its elements defining a one-to-one mapping ϕ from M to some open subset in $\mathbb{R}^d$. Then M is called a C^∞ differentiable manifold, or smooth manifold, if two arbitrary maps $\phi, \psi \in A$ satisfy the condition that $\psi \circ \phi^{-1}$ is a C^∞ diffeomorphism.[3] This is conceptually illustrated in Figure B.1. For the remainder of this chapter, manifold M refers exclusively to a smooth manifold.

B.2 Tangent Spaces

- *Tangent space*: Tangent space T_pM at a point p in the manifold M is basically the vector space spanned by tangent vectors of all coordinate curves at p. T_pM is important as

[1] A Hausdorff space is a topological space in which distinct points have disjoint neighborhoods. This is consistent with our intuition that different points should be treated as different objects.

[2] Being locally Euclidean means that each point $p \in M$ has a neighborhood homeomorphic (but does not have to be isomorphic) to an d-ball in $\mathbb{R}^n$.

[3] C^∞ diffeomorphism means that $\psi \circ \phi^{-1}$ and its inverse $\phi \circ \psi^{-1}$ are both C^∞ (infinitely many times differentiable).

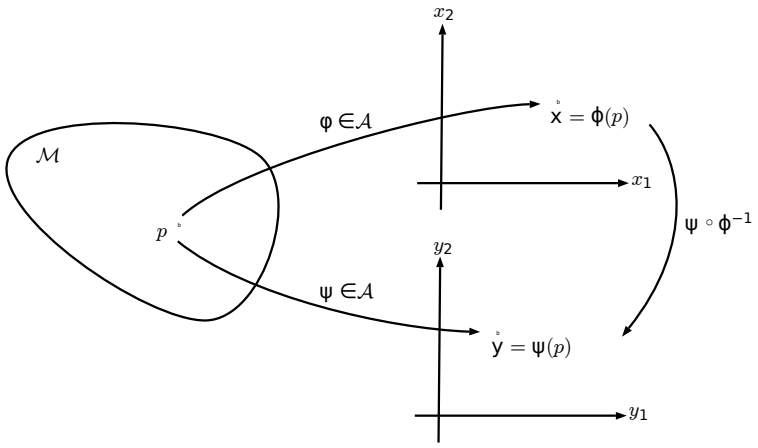


Figure B.1: Illustration of a smooth 2-manifold. Here two coordinate systems are given, and $\psi \circ \phi^{-1}$ is a C^∞ diffeomorphism.

it gives a local linearization of M at p, so that the study of the structure of M can be made easier.

Let $\{E_i^p\}$ be a local coordinate system at point $p \in M$, and $\gamma(t)$ be a parameterized curve passing through p (namely $p = \gamma(a)$ for some a). The tangent vector of $\gamma(t)$ at p can be expressed as $\dot{\gamma}(a) = \dot{\gamma}^i(a) \left(\frac{\partial}{\partial E_i}\right)_p$[4], in which $\dot{\gamma}^i(a) \equiv \frac{d}{dt}\gamma^i(t)|_{t=a}$, and the operator $\left(\frac{\partial}{\partial E_i}\right)_p$ gives the tangent vector at p of the i^{th} coordinate curve (that is $\left(\frac{\partial}{\partial E_i}\right)_p$ form a set of basis for T_pM). Consider all curves passing through p, and the tangent space at p is the set of all tangent vectors corresponding to these curves:

$$T_pM = \left(c^i \left(\frac{\partial}{\partial E_i}\right)_p \right) [c^1, \ldots, c^d] \in \mathbb{R}^d$$

This forms a linear space spanned by $\left(\frac{\partial}{\partial E_i}\right)$, and since they are linearly independent, the

[4] As a convention, in this chapter the Einstein notation is used to represent summation. Basically, indices that are both superscripted and subscripted are summed over, omitting the summation sign $\sum$. For example, $x^i y_i$ stands for $\sum_i x^i y_i$.

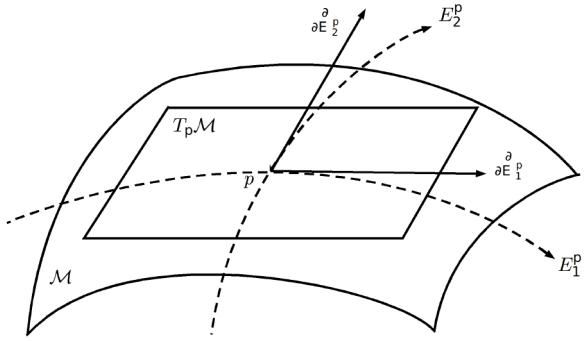


Figure B.2: Illustration of T_pM .

dimension of T_pM is d (the same as the dimension of M). Tangent space is illustrated in Figure B.2.

- *Vector fields and tangent bundles*: A vector field is a mapping $X : p \to X_p$ where $p \in M$ and $X_p \in T_pM$.

A tangent bundle TM is the disjoint union of tangent spaces T_pM for all $p \in M$: $TM = \bigsqcup_{p \in M} T_pM$.[5]

B.3 Riemannian Metric

Riemannian metric defines the inner products on tangent spaces. Formally, a Riemannian metric g on a manifold M is a 2-tensor field g satisfying the following properties:

1. *Symmetric*: For all $p \in M$ and all tangent vectors $X, Y \in T_pM$, $g_p(X_p, Y_p) = g_p(Y_p, X_p)$.

2. *Positive definite*: For all $p \in M$ and all tangent vectors $X \in T_pM$, $g_p(X_p, X_p) \geq 0$

[5]Disjoint union means that elements in TM are indexed by their base points p.

A smooth manifold equipped with Riemannian metrics is called Riemannian manifold.

It is often convenient to express the metric tensor g as a symmetric, positive definite matrix $G = [g_{ij}]$, where g_{ij} are determined by $g_{ij} = \langle \partial_i, \partial_j \rangle$ (∂_i , $\frac{\partial}{\partial E_i}$), namely the inner products between coordinate frames. Let two vector fields $X, Y \in T_pM$ be expressed as $X = X^i(\partial_i)_p$, $Y = Y^i(\partial_i)_p$, then $g_p(X, Y) = \langle X, Y \rangle = g_{ij}(p)X^iY^i = \boldsymbol{x}^T G_p \boldsymbol{y}$, where $\boldsymbol{x} = [X^i]$, $\boldsymbol{y} = [Y^i]$. A trivial example of G is the identity matrix, which recovers the dot product for Euclidean geometry.

Given a g, the angle between $X, Y \in T_pM$ is given by $\cos^{-1} \frac{g_p(X,Y)}{[g_p(X,X)]^{1/2}[g_p(Y,Y)]^{1/2}}$.

B.4 Connections

- *Motivation*: In order to study the differential properties of Riemann manifolds, obviously we have to define derivatives first. For example, if we want to compute the directional derivative of a vector field X along a curve $\gamma(t)$, a natural thought would be to compute $\lim_{\varepsilon \to 0} \frac{X(\gamma(t+\varepsilon)) - X(\gamma(t))}{\varepsilon}$. While this works for Euclidean space, it does not make sense for a curved space! The trouble is that $X(\gamma(t+\varepsilon))$ and $X(\gamma(t))$ lie in their own tangent spaces $T_{\gamma(t+\varepsilon)}M$ and $T_{\gamma(t)}M$ which has nothing to do with each other [6], therefore $X(\gamma(t+\varepsilon)) - X(\gamma(t))$ is meaningless. In order to enable differential operations on a Riemannian manifold, we have to set up a "connection" between two nearby tangent spaces T_pM , T_qM , that is to transport a vector field in T_pM to T_qM first, then compute deriva-

[6] Although a global coordinate system $\{E_i\}$ can be defined for M , the sets of basis for two different tangent spaces T_pM , T_qM , given by $\frac{\partial}{\partial E_i^p}$ and $\frac{\partial}{\partial E_i^q}$ respectively, are different.

tives in the same space. It should be noted that there is no canonical way of setting up a connection, and different connections result in different geometric structures. However, they have to satisfy some axioms as we will see shortly. What is more, it is important that the definition of connection must be invariant to different choices of coordinate systems, since they can be arbitrarily specified but the intrinsic geometric properties should not subject to the choice of coordinates.

- *Parallel transportation*: When p and q are very close so that the second order difference can be ignored, the simplest way to transport $X(p) \in T_pM$ to T_qM and set up a connection is to use a linear map $\Pi_{p,q} : T_pM \to T_qM$. The image of X after transportation is $\Pi_{p,q}(X(p))$, and we may now compute the derivative of X as $\lim_{dE\to 0} \frac{X(q)-\Pi_{p,q}(X(p))}{|dE|}$, where $q = p + dE$ and dE the difference between their coordinates, since they are now in the same tangent space. Let us assume that the difference $X(q) - \Pi_{p,q}(X(p))$ is in the form $X(q) - \Pi_{p,q}(X(p)) = dE^i(\Gamma_{ij}^k)_p(\partial_k)_q$, that is it can be expressed as a linear combination of dE^i. Γ_{ij}^k are named Christoffel symbols (or connection coefficients), whose definition and meaning will become clear shortly. A connection constructed in this way is named *affine connection* and is simply called connection hereafter with no danger of confusion, although other forms of connections exist. See Figure B.3 for an illustration.

Based on this construction, we may now define parallelism on M. A vector field X is said to be parallel along a curve $\gamma(t)$ if $X(t + dt) = \Pi_{\gamma(t),\gamma(t+dt)}(X(t))$ for all t on which $\gamma(t)$ is defined. That is to say as X move along $\gamma(t)$, its derivative remains zero all the

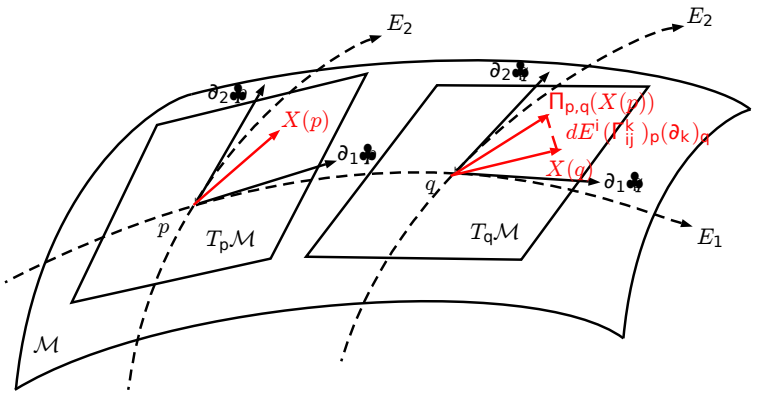


Figure B.3: Illustration of affine connection.

time, and such a transportation is called parallel transportation.[7]

We note again that the choice of Π and Γ^k_{ij} is not unique, and there is no canonical structure defined for M. Therefore it is only meaningful to say some geometric property holds *with respect to a specific connection*.

- *Formal definition*: Let $\{E^p_i\}$ be a local coordinate frame defined for T_pM [8], and a vector field $X_p = X^i_p E^p_i$ defined at point p. Define $X_p f$, $\frac{d}{dt} f(p + tX_p)|_{t=0} = X^i_p \frac{\partial f}{\partial E_i}|_p$ to be the local directional derivative of f in the direction of X_p, and $Xf(p)$, $X_p f$ be a function of p. Also define $(fX)_p$, $f(p)X_p$, which is still a vector field. An affine connection in M is a map $\nabla : TM \times TM \to TM$ satisfying axioms

1. $\nabla_{fX_1+gX_2} Y = f\nabla_{X_1} Y + g\nabla_{X_2} Y, \quad \forall f, g \in C^\infty(M)$

2. $\nabla_X (aY_1 + bY_2) = a\nabla_X Y_1 + b\nabla_X Y_2$

3. $\nabla_X (fY) = f\nabla_X Y + (Xf)Y, \quad \forall f \in C^\infty(M)$

[7] Do not confuse the parallelism defined here with the commonly used concept defined for Euclidean geometry. Here parallelism only means a linear relation between X_p and X_q.

[8] Usually $\{E^p_i\}$ are given by $\frac{\partial}{\partial x^i}$ of local coordinates $\{x_i\}$ at $p \in M$.

where $\nabla_X Y$ (as the result of the map) is called the covariant derivative of Y in the direction of X. The word "covariant" implies "coordinate-invariant", namely the definition is adaptive to the choice of coordinates. In fact, $\nabla_X Y$ is just the directional derivative defined for a Riemannian manifold $\lim_{dE\to 0} \frac{X(q)-\Pi_{p,q}(X(p))}{|dE|}$ where q is infinitesimally close to p in the direction given by X. Let $X = X^i E_i$, $Y = Y^j E_j$, it can be shown that $\nabla_X Y$ has the following form:

$$\nabla_X Y = (XY^k + X^i Y^j \Gamma_{ij}^k) E_k = \sum_{k=1}^{n} \left(\sum_{i=1}^{n} X^i \frac{\partial y^k}{\partial x_i} + \sum_{i,j} \Gamma_{i,j}^k X^i Y^j \right) E_k$$

in which the Christoffel symbols are defined as $\nabla_{E_i} E_j = \Gamma_{ij}^k E_k$, and there are d^3 such functions. Γ_{ij}^k measures the change of the basis vector E_j as it moves in the direction of E_i. Although the actual value of Γ_{ij}^k depends on p, the functional form of $\nabla_X Y$ remains invariant under different coordinate systems.

Finally, it should be noted that the affine connection and $\nabla_X Y$ are indeed equivalent. Given a connection the corresponding $\nabla_X Y$ is also determined, and vice versa. Therefore, defining $\nabla_X Y$ in the way given above is equivalent to defining a connection.

- *Levi-Civita connection*: It can be shown that many possible connections exist on a manifold, in other words there are many ways to define rules of derivatives as long as they satisfy axioms 1-3 above. Among all these connections, there is one connection that additionally satisfies the following conditions:

4. $Z\langle X, Y\rangle = \langle \nabla_Z X, Y\rangle + \langle X, \nabla_Z Y\rangle, \quad \forall X, Y, Z \in TM$

5. $\nabla_X Y - \nabla_Y X = [X, Y]$

Here $\langle \cdot \rangle$ is the inner product with respect to the Riemannian metric, $[X, Y]$ is a vector field defined as $(X^j \partial_j Y^i - Y^j \partial_j X^i)\partial_i$ where $\partial_i \equiv \frac{\partial}{\partial E_i}$. Property 4 guarantees that the connection is compatible with the Riemannian metric, that is the inner product between two vector fields before and after parallel transportation remains constant; Property 5 guarantees that the torsion tensor, defined as $\tau(X, Y) = \nabla_X Y - \nabla_Y X - [X, Y]$, vanishes with respect to this connection.

This connection is called Levi-Civita connection [9], and can be completely determined by the Riemannian metric as follows:

$$\Gamma_{ij}^k = \frac{1}{2} g^{kl} (\partial_j g_{il} + \partial_i g_{lj} - \partial_l g_{ij})$$

where g^{kl} are the entries of the dual metric tensor (inverse of G). It can be shown that the Levi-Civita connection is the unique connection on a Riemannian manifold that satisfies conditions 1-5, and this is known as the *Fundamental Lemma of Riemannian Geometry*. It is also easy to see that this connection is symmetric: $\Gamma_{ij}^k = \Gamma_{ji}^k$.

Due to the uniqueness of the Levi-Civita connection, it is often the only connection discussed in many textbooks whereas other types of connections are not even mentioned. However, the analysis technique of information geometry developed by Amari, introduced in Section 2.2, studies the dual α-connection instead. This is because the Levi-Civita con-

[9] Although not discovered by Riemann, this connection is also called the Riemannian connection or metric connection.

nection often makes the analysis of statistical manifolds intractable.

B.5 Some Geometric Objects and Properties

The construction of Riemann metrics and connections makes the definition of many geometric objects and the study of their properties possible.

- *Geodesics*: Let $\gamma(t) : [a, b] \to M$ be a parameterized curve in a Riemannian manifold M. The velocity of the curve is defined as the vector field $\dot{\gamma}(t) \in T_{\gamma(t)}M$, and the speed of the curve is given by the length of the velocity vector field: $\int_a^b \left\| \frac{d\gamma}{dt} \right\| dt = \int_a^b \sqrt{g_{ij}\dot{\gamma}^i\dot{\gamma}^j}\, dt$, where $\dot{\gamma}^i(a)$, $\frac{d}{dt}\gamma^i(t)|_{t=a}$. Given a connection ∇, the acceleration of the curve is defined as the vector field $\nabla_{\dot{\gamma}(t)}\dot{\gamma}(t)$, and a curve is called a geodesic with respect to ∇ if its acceleration is zero. Based on the definition of $\nabla_X Y$, it can be shown that $\gamma(t) = [\gamma^1(t), \ldots, \gamma^d(t)]$ is a geodesic if and only if it satisfies the ordinary differential equation (ODE): $\ddot{\gamma}^k(t) + \gamma^i(t)\gamma^j(t)\Gamma_{ij}^k(\gamma(t)) = 0$

Since connections are not unique, there are many possible geodesics between two points $\gamma(a)$ and $\gamma(b)$. However among all of them, it can be shown that the geodesic with respect to the Levi-Civita connection has the minimum length, due to its symmetric nature. Therefore, this geodesic is often referred to as the "shortest path" between two (local) points on a manifold.

- *The exponential map*: It can be shown that given an initial point $p \in M$ and velocity vector $V \in T_pM$, a unique geodesic $\gamma_V(t)$ can be determined.[10] This implicitly determines

[10] In fact the specification of $p \in M$ is not necessary since it can be determined by V.

a map from TM to the set of geodesics in M, and this map is called exponential map.

More precisely, let V, $\{V \in TM : \gamma_V$ is defined on an interval containing $[0, 1]\}$ then the exponential map $\exp : V \to M$ is defined as $\exp(V) = \gamma_V(1)$. It can be shown that the exponential map is smooth and $\gamma_V(t) = \exp(tV.)$

- *Curvature*: Let ∇ be a connection in M, then for vector fields $X, Y, Z \in TM$, the Riemannian curvature tensor is a map $R : TM \times TM \times TM \to TM$ defined as

$$R(X, Y)Z = \nabla_X(\nabla_Y Z) - \nabla_Y(\nabla_X Z) - \nabla_{[X,Y]}Z$$

Intuitively, the curvature tensor measures the degree of M being locally *isometric* to Euclidean space. When curvature vanishes at some point p on a manifold, the manifold is locally isometric to Euclidean space, that is p has a neighborhood that is isometric to an open set in $\mathbb{R}^d$ using Euclidean metric. If curvature vanishes everywhere in M, then M is said to be a *flat Riemannian manifold* (with respect to ∇).[11]

- *Affine coordinate system*: Let $\{E_i\}$ be a coordinate system of M. If the d basis vector fields $\partial_i = \frac{\partial}{\partial E_i}(i = 1, \ldots, d)$ are all parallel on M with respect to connection ∇, then $\{E_i\}$ is called an affine system (with respect to ∇). This is equivalent to $\nabla_{\partial_i}\partial_j = 0, \forall i, j$, or $\Gamma_{ij}^k = 0, \forall i, j, k$. Therefore an affine coordinate system exists if and only if M is flat.

A second affine coordinate system $\{F_i\}$ on M exists if and only if there exists an affine

[11] Roughly speaking, flat manifolds are those that can be built by gluing together tiny pieces of d-dimensional "rectangles" everywhere. For example cylinder, torus, Klein bottle are all flat, but sphere is not.

transformation: $E_p = AF_p + B,\ \forall p \in M$, where E_p, F_p are the coordinates given by $\{E_i\}$ and $\{F_i\}$ at point p respectively, $A \in \mathbb{R}^{d \times d}$, $B \in \mathbb{R}^d$.

Appendix C

f-Divergence and Bregman Divergence

Although a lot of divergence functions have been proposed in history (see for example 206–208 for general introductions), it is quite interesting that many of these independently proposed divergences share some common structures and can be viewed from a more abstract perspective. The f-divergence and Bregman divergence are two important general divergence classes that captures such structures, and they subsume most commonly used divergences. Therefore, the study of the two general classes is very important. Other general divergence classes, for example the Φ-divergence 209, also exist.

C.1 f-Divergence

- The f-divergence was first proposed by Csiszár 122 and later by Ali and Silvey 210 independently. Let $f : (0, \infty) \to \mathbb{R}$ be a convex function with $f(1) = 0$, and p, q be two

probability distributions on a set X. The f-divergence is defined as (consider only discrete case here)

$$D_f(p, q) = \sum_{x \in X} p(x) f\left(\frac{q(x)}{p(x)}\right)$$

In some literature, D_f is alternatively defined as $D_f(p, q) = \sum_{x \in X} q(x) \frac{p(x)}{q(x)}$ but the difference is not important. Also, sometimes extra conditions on f are imposed for regularity: $f'(1) = 0, f''(1) = 1$

Intuitively, the f-divergence encodes the expectation of a "contrast" $f\left(\frac{q(x)}{p(x)}\right)$ between p and q defined by f, which measures the "distance" between the two distributions.

- Some examples of f-divergences are given in Table C.1:

Table C.1: Examples of f-Divergences.

$f(x)$	$D_f(p, q)$	Name
$x \log x$	$\sum_x q(x) \log \frac{q(x)}{p(x)}$	KL divergence
$-\log x$	$\sum_x p(x) \log \frac{p(x)}{q(x)}$	
$\lvert x - 1 \rvert$	$\sum_x \lvert p(x) - q(x) \rvert$	Total variation distance
$(x-1)^2$	$\sum_x \frac{(p(x)-q(x))^2}{p(x)}$	χ^2 divergence
$x^2 - 1$	$\sum_x \frac{q(x)^2 - p(x)^2}{p(x)}$	
$(\sqrt{x} - 1)^2$	$\sum_x \left(\sqrt{p(x)} - \sqrt{q(x)}\right)^2$	Hellinger divergence
$1 - \sqrt{x}$	$\sum_x p(x) - \sqrt{p(x)q(x)}$	
$x \log \frac{2x}{x+1} + \log \frac{2}{x+1}$	$KL(p, \frac{p+q}{2}) + KL(q, \frac{p+q}{2})$	Jensen-Shannon divergence

- Let $f^*(x) = xf\left(\frac{1}{x}\right) + c(x-1), \forall x > 0$ where $c \in \mathbb{R}$ is a constant. It can be easily verified that f^* is also convex and $f^*(1) = 0$ The dual f-divergence is defined by $D_f^*(p, q) \triangleq D_{f^*}(p, q)$, and the equality $D_f^*(p, q) = D_f(q, p)$ holds.

An f-divergence is said to be symmetric if it is equal to its dual.

- The f-divergence is decomposable, that is $D_f(p, q) = \sum_{x \in X} d(p(x), q(x))$ where $d(p(x), q(x)) \triangleq p(x) f\left(\frac{q(x)}{p(x)}\right)$.

- The f-divergence satisfies the invariance property. That is, under an invertible mapping $m : X \to Y$, the f-divergence between two distributions $p_1(x), p_2(x)$ before and after the mapping remains the same. To see this, let $q_1(y) = q_1(m(x)) = p_1(x)|M(x)|^{-1}$, $q_2(y) = q_2(m(x)) = p_2(x)|M(x)|^{-1}$ where $|M|$ is the determinant of the Jacobian of the function m. Then

$$\begin{aligned} D_f(q_1, q_2) &= \int_{y \in Y} q_1(y) f\left(\frac{q_2(y)}{q_1(y)}\right) dy \\ &= \int_{x \in X} p_1(x)|M(x)|^{-1} f\left(\frac{p_2(x)}{p_1(x)}\right) |M(x)| dx \\ &= D_f(p_1, p_2) \end{aligned}$$

In fact, f-divergence is the only statistical invariant divergence, see for example 211.

- The f-divergence satisfies the information monotonicity property. Consider discrete distributions on X with m elements. Let $X = X_1 \cup \ldots \cup X_k$ be an arbitrary partition of X into $k < m$ bins, and let $\bar{p}_1, \bar{p}_2$ be two distributions defined on $\{X_1, \ldots, X_k\}$. Then it can be shown using Jensen's inequality that $D_f(\bar{p}_1, \bar{p}_2) \leq D_f(p_1, p_2)$.

The information monotonicity property is desirable since if we coarse-grain the support of the distribution, it should become more difficult to tell the difference between two distributions. In the extreme case where $k = 1$, then $D_f(\bar{p}_1, \bar{p}_2) = 0$. In fact, the f-

divergence is the only decomposable divergence that satisfy information monotonicity, see for example 125.

Many other properties of f-divergences (for example the divergence bounds and inequalities) and its application in statistics can be found Bounds in, for example, 212, 213 and the references therein.

C.1.1 α-Divergences

An important subclass of f-divergences is the α-divergence. In history, researchers have proposed different versions of α-divergences which are very similar to each other, and this makes the name sometimes confusing. In this section, we give a brief summary of α-divergences and its variations.

- Amari α-Divergence[1]:

$$D_f^{(\alpha)}(p, q) = \frac{1}{\alpha(1-\alpha)}\left(1 - \sum_i p_i^{\alpha} q_i^{1-\alpha}\right)$$

When $\alpha = 1$ or 0, $D_f^{(\alpha)}(p, q)$ converges to $KL(p, q)$ and $KL(q, p)$ respectively. The corre-

[1]The definition here is equivalent to Eq. 2.15 if we do a simple transformation of α.

sponding $f(x)$ is

$$f(x) = \begin{cases} \frac{4}{1-\alpha^2}\left(1 - x^{\frac{1+\alpha}{2}}\right) & \alpha \neq \pm 1 \\ x \log x & \alpha = 1 \\ -\log x & \alpha = -1 \end{cases}$$

The Amari α-divergence is also known as the Havrda-Charvát divergence or Csiszár's f-divergence of order α, and is perhaps the most commonly used form of α-divergences. The prototype of this divergence was proposed by Chernoff 214, and has been proposed and extended by several researchers independently later, see for example 207, 208 for a survey. The Power divergence proposed by 215 is also equivalent to the Amari α-divergence.

- Tsallis α-Divergence216, 217:

$$D_f^{(\alpha)}(p, q) = \frac{1}{(\alpha - 1)}\left(\sum_i p_i^{\alpha} q_i^{1-\alpha} - 1\right)$$

This is just a rescaled version of the Amari α-divergence. However, unlike the Amari α-divergence, the Tsallis α-divergence is derived from the Tsallis relative entropy (also known as the generalized KL-divergence, in which the $\log$ in the normal KL-divergence is replaced by the Tsallis q-$\log$).

- Rényi α-Divergence 218:

$$D^{(\alpha)}(p, q) = \frac{1}{(\alpha - 1)} \log\left(\sum_i p_i^{\alpha} q_i^{1-\alpha}\right)$$

Note that the Rényi α-Divergence does *not* belong to the f-divergence family! However, it is a function of some members of the f-divergence (Power divergence, Hellinger distance, total-variation distance etc.) and appears similar to other α-divergences, therefore we also include it here for comprehensiveness. See 219 for a survey of the Rényi α-divergence properties and its connections with f-divergences.

C.2 Bregman Divergence

- The Bregman divergence, proposed by Lev M. Bregman 124, is defined as

$$D_G(\boldsymbol{x}, \boldsymbol{y}) = G(\boldsymbol{x}) - G(\boldsymbol{y}) - \nabla G(\boldsymbol{y})^{\mathsf{T}}(\boldsymbol{x} - \boldsymbol{y})$$

where G is a strictly convex function, $\boldsymbol{x}, \boldsymbol{y}$ are usually scalars or vectors. However, Bregman divergence can also be used to reflect the "nearness" between matrices, in which case $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^{d \times d}$. See for example 220–222. In the case of real, symmetric matrices, the Bregman divergence is similarly defined as

$$D_G(\boldsymbol{x}, \boldsymbol{y}) = G(\boldsymbol{x}) - G(\boldsymbol{y}) - \operatorname{tr}((\nabla G(\boldsymbol{y})^{\mathsf{T}})(\boldsymbol{x} - \boldsymbol{y}))$$

- Some examples of Bregman divergences are given in Table C.2:

Table C.2: Examples of Bregman Divergences.

Domain	$G(\boldsymbol{x})$	$D_G(\boldsymbol{x}, \boldsymbol{y})$	Name
$(d-1)$-simplex	$\sum_{i=1}^{d} x_i \log x_i$	$\sum_{i=1}^{d} x_i \log \frac{x_i}{y_i}$	KL divergence
$\mathbb{R}^d$	$\|\boldsymbol{x}\|^2$	$\|\boldsymbol{x}-\boldsymbol{y}\|^2$	Squared Euclidean distance
$\mathbb{R}^d$	$\boldsymbol{x}^T A \boldsymbol{x}$	$(\boldsymbol{x}-\boldsymbol{y})^T A (\boldsymbol{x}-\boldsymbol{y})$	Mahalanobis distance
$\mathbb{R}_+$	$-\log x$	$\frac{x}{y} - \log \frac{x}{y} - 1$	Itakura-Saito distance
$\mathbb{R}$	e^x	$e^x - e^y - (x-y)e^y$	N/A
$\mathbb{R}^{d\times d}$, symmetric	$\sum_{i=1}^{d} \lambda_i \log \lambda_i$	$\mathrm{tr}(\boldsymbol{x}\log\boldsymbol{x} - \boldsymbol{x}\log\boldsymbol{y})$	von Neumann divergence
$\mathbb{R}^{d\times d}$, symmetric	$-\sum_{i=1}^{d} \log \lambda_i$	$\mathrm{tr}(\boldsymbol{x}\boldsymbol{y}^{-1}) - \log\det(\boldsymbol{x}\boldsymbol{y}^{-1}) - d$	LogDet divergence

- Bregman divergence has a clear graphical interpretation, as shown in Figure C.1 for scalar variable case. It is just the gap between $G(\boldsymbol{x})$ and its first-order approximation pivoted at $\boldsymbol{y}$.

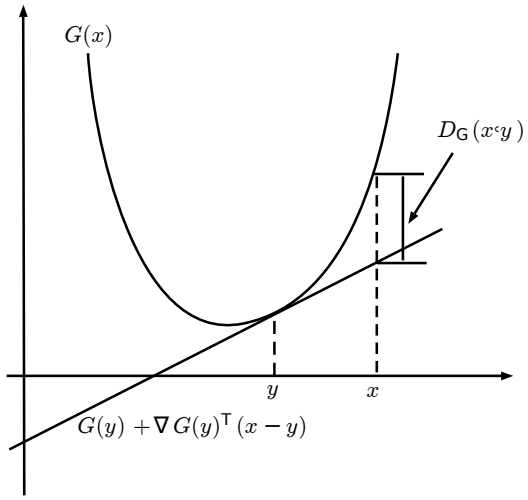


Figure C.1: Illustration of the Bregman divergence.

- The Fenchel-Young inequality (see Section A.2) can be used to defined the Bregman

divergence in its dual form:

$$D_G(\boldsymbol{x}, \boldsymbol{y}) = G(\boldsymbol{x}) + H(\boldsymbol{y}) - \boldsymbol{x}^T\boldsymbol{y} \tag{C.1}$$

where H is the convex conjugate of G. The Fenchel-Young inequality guarantees that $G(\boldsymbol{x}) + H(\boldsymbol{y}) - \boldsymbol{x}^T\boldsymbol{y} \geq 0$

- Bregman divergences satisfy the following properties:

- *Linearity*:

$$D_{G_1+\lambda G_2}(\boldsymbol{x}, \boldsymbol{y}) = D_{G_1}(\boldsymbol{x}, \boldsymbol{y}) + \lambda D_{G_2}(\boldsymbol{x}, \boldsymbol{y})$$

- *Generalized law of cosines*:

$$D_G(\boldsymbol{x}, \boldsymbol{y}) + D_G(\boldsymbol{y}, \boldsymbol{z}) = D_G(\boldsymbol{x}, \boldsymbol{z}) + (\boldsymbol{x} - \boldsymbol{y})\,(\nabla G(\boldsymbol{z}) - \nabla G(\boldsymbol{y}))$$

- *Generalized Pythagorean theorem*:

Let $\boldsymbol{z}$ be the Bregman projection of $\boldsymbol{y}$ onto a convex set Ω: $\boldsymbol{z} = \underset{\boldsymbol{z}^0 \in \Omega}{\operatorname{argmin}} D_G(\boldsymbol{z}^0, \boldsymbol{y})$, then $\forall \boldsymbol{x} \in \Omega$

$$D_G(\boldsymbol{x}, \boldsymbol{y}) \geq D_G(\boldsymbol{x}, \boldsymbol{z}) + D_G(\boldsymbol{z}, \boldsymbol{y})$$

The equality holds when Ω is an affine set.

- Relationship with Exponential Families

There is a one-to-one correspondence between exponential families and Bregman divergence (see for example 223, Theorem 4). To see this, consider an exponential family of

the form

$$p(\boldsymbol{x};\boldsymbol{\theta}) = p_0(\boldsymbol{x})\exp\{\boldsymbol{\theta}^T\boldsymbol{x} - G(\boldsymbol{\theta})\}$$

where $G(\boldsymbol{\theta})$ is the partition function. Let $h = \nabla H$ where H is the convex conjugate of G, then $\boldsymbol{x} = \nabla G(h(\boldsymbol{x}))$. Therefore

$$\begin{aligned} p(\boldsymbol{x};\boldsymbol{\theta}) &= \tilde{p}_0(\boldsymbol{x})\exp\{-G(\boldsymbol{\theta}) + G(h(\boldsymbol{x})) + \nabla G(h(\boldsymbol{x}))^T(\boldsymbol{\theta} - h(\boldsymbol{x}))\} \\ &= \tilde{p}_0(\boldsymbol{x})\exp\{-D_G(\boldsymbol{\theta}, h(\boldsymbol{x}))\} \end{aligned}$$

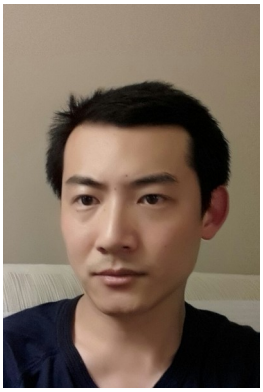
where $\tilde{p}_0(\boldsymbol{x}) = p_0(\boldsymbol{x})\exp\{G(h(\boldsymbol{x})) - \boldsymbol{x}^T h(\boldsymbol{x})\}$.

Some examples are given in Table C.3.

Table C.3: Examples of Correspondence Between Exponential Families and Bregman Divergences.

$G(\boldsymbol{x})$	$D_G(\boldsymbol{x}, \boldsymbol{y})$	Distribution
$\frac{1}{2}\|\boldsymbol{x}\|_2^2$	$\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2$	Gaussian
e^x	$e^x - e^y - (x - y)e^y$	Poisson
$\log(1 + \exp(x))$	$\log\frac{1+e^x}{1+e^y} - \frac{(x-y)e^x}{1+e^y}$	Bernoulli

Vita



Yuan Cao was born and grew up in Shanghai, China. He received his B.S in automation in 2005 and M.S in pattern recognition in 2008, both from Shanghai Jiaotong University, China, and graduated with honor. From 2008 to 2009, he worked at General Electric, China. In September 2009, he was enrolled in the Electrical Engineering Ph.D. program at Johns Hopkins University, working with Prof. Sanjeev Khudanpur at the Center for Language and Speech Processing. His research focused on discriminative training and machine learning with applications to natural language processing problems.

Starting in October 2015, Yuan Cao joined Google Translate team in Mountain View, CA, working on machine translation models that helps to improve the Google Translate products.

Bibliography

[1] T. Jebara, *Machine Learning: Discriminative and Generative*. New York: Kluwer Academic Publishers, 2004.

[2] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifers: A comparison of logistic regression and naive bayes," in *Proceedings of NIPS*, 2002.

[3] G. Bouchard and B. Triggs, "The trade-off between generative and discriminative classifiers," in *Proceedings of International Conference on Computational Statistics*, 2004.

[4] J.-H. Xue and D. M. Titterington, "Comment on "on discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes"," 2008.

[5] P. Liang and M. I. Jordan, "An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators," in *Proceedings of ICML*, 2008.

[6] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. Cambridge, Massachusetts: The MIT Press, 2009.

BIBLIOGRAPHY

[7] T. Jaakkola, M. Meila, and T. Jebara, "Maximum entropy discrimination," in *Proceedings of NIPS*, 1999.

[8] T. P. Minka, "Discriminative models, not discriminative training," Microsoft Research, MSR-TR-2005-144, Tech. Rep., 2005.

[9] J. A. Lasserre, C. M. Bishop, and T. P. Minka, "Principled hybrids of generative and discriminative models," in *Proceedings of CVPR*, 2006.

[10] C. M. Bishop and J. A. Lasserre, "Generative or discriminative? getting the best of both worlds," *Bayesian Statistics*, vol. 3, 2007.

[11] P. Koehn, *Statistical Machine Translation*. Cambridge University Press, 2010.

[12] K. Papineni, S. Roukos, T. Ward, and W. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of ACL*, 2002.

[13] R. Rosenfeld, "A maximum entropy approach to adaptive statistical language modelling," *Computer Speech & Language*, vol. 10(3), 1996.

[14] A. L. Berger, S. A. D. Pietra, and V. J. D. Pietra, "A maximum entropy approach to natural language processing," *Computational Linguistics*, 1996.

[15] A. Ratnaparkhi, "A maximum entropy model for part-of-speech tagging," in *Proceedings of EMNLP*, 1996.

[16] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, 1958.

BIBLIOGRAPHY

[17] M. Collins, "Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms," in *Proceedings of EMNLP*, 2002.

[18] B. Taskar, C. Guestrin, and D. Kolle, "Max-margin markov networks," in *Proceedings of NIPS*, 2003.

[19] G. Bakir, T. Hofmann, B. Schölkopf, A. J. Smola, B. Taskar, and S. Vishwanathan, *Predicting Structured Data*. Cambridge, Massachusetts: The MIT Press, 2007.

[20] H. Yu, L. Huang, H. Mi, and K. Zhao, "Max-violation perceptron and forced decoding for scalable mt training," in *Proceedings of EMNLP*, 2013.

[21] K. Zhao, L. Huang, H. Mi, and A. Ittycheriah, "Hierarchical mt training using max-violation perceptron," in *Proceedings of ACL*, 2014.

[22] K. Crammer and Y. Singer, "Ultraconservative online algorithms for multiclass problems," *Journal of Machine Learning Research*, vol. 3, 2003.

[23] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-agreesive algorithms," *Journal of Machine Learning Research*, vol. 7, 2006.

[24] D. Chiang, Y. Marton, and P. Resnik, "Online large-margin training of syntactic and structural translation features," 2008.

[25] C. Cherry and G. Foster, "Batch tuning strategies for statistical machine translation," in *Proceedings of HLT-NAACL*, 2012.

BIBLIOGRAPHY

[26] K. Crammer, A. Kulesza, and M. Dredze, "Adaptive regularization of weight vectors," in *Proceedings of NIPS*, 2009.

[27] D. Chiang, "Hope and fear for discriminative training of statistical translation models," *Journal of Machine Learning Research*, vol. 13, 2012.

[28] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, 2011.

[29] F. Och, "Minimum error rate training for statistical machine translation," in *Proceedings of ACL*, 2003.

[30] M. Hopkins and J. May, "Tuning as ranking," in *Proceedings of EMNLP*, 2011.

[31] K. Gimpel and N. Smith, "Structured ramp loss minimization for machine translation," in *Proceedings of HLT-NAACL*, 2012.

[32] I. Tsochantaridis, T. Hoffman, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in *Proceedings of ICML*, 2004.

[33] D. Smith and J. Eisner, "Minimum risk annealing for training log-linear models," in *Proceedings of ACL*, 2006.

[34] Z. Li and J. Eisner, "Minimum imputed risk: unsupervised discriminative training for machine translation," in *Proceedings of EMNLP*, 2011.

BIBLIOGRAPHY

[35] L. Shen and A. K. Joshi, "An svm based voting algorithm with application to parse reranking," 2003.

[36] B. Taskar, D. Klein, M. Collins, and D. Koller, "Max-margin parsing," 2004.

[37] E. Charniak and M. Johnson, "Coarse-to-fine n-best parsing and maxent discriminative reranking," in *Proceedings of ACL*, 2005.

[38] M. Collins and T. Koo, "Discriminative reranking for natural language parsing," *Computational Linguistics*, 2005.

[39] B. Roark, M. Saraclar, and M. Collins, "Discriminative n-gram language modeling," *Computer Speech and Language*, 2007.

[40] Z. Li and S. Khudanpur, "Large-scale discriminative n-gram language models for statistical machine translation," in *Proceedings of AMTA*, 2008.

[41] C. Cherry and C. Quirk, "Discriminative, syntactic language modeling through latent svms," in *Proceedings of AMTA*, 2008.

[42] P. Xu, D. Karakos, and S. Khudanpur, "Self-supervised discriminative training of statistical language models," 2009.

[43] L. Shen, A. Sarkar, and F. Och, "Discriminative reranking for machine translation," 2004.

[44] P. Liang, A. Bouchard-Côté, D. Klein, and B. Taskar, "An end-to-end discriminative approach to machine translation," in *Proceedings of ACL*, 2006.

BIBLIOGRAPHY

[45] T. Watanabe, J. Suzuki, H. Tsukada, and H. Isozaki, "Online large-margin training for statistical machine translation," 2007.

[46] S. Green, S. Wang, D. Cer, and C. D. Manning, "Fast and adaptive online training of feature-rich translation models," 2013.

[47] R. Moore, "A discriminative framework for bilingual word alignment," 2005.

[48] B. Taskar, S. Lacoste-Julien, and D. Klein, "A discriminative matching approach to word alignment," 2005.

[49] P. Blunsom and T. Cohn, "Discriminative word alignment with conditional random fields," 2006.

[50] K. Toutanova and C. D. Manning, "Enriching the knowledge sources used in a maximum entropy part-of-speech tagger," in *Proceedings of EMNLP*, 2000.

[51] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *Proceedings of HLT-NAACL*, 2003.

[52] Y. Altun, "Discriminative methods for label sequence learning," Ph.D. dissertation, Brown University, 2005.

[53] J. Jagarlamudi and H. D. III, "Low-dimensional discriminative reranking," in *Proceedings of HLT-NAACL*, 2012.

[54] F. Sha and F. Pereira, "Shallow parsing with conditional random fields," in *Proceedings of HLT-NAACL*, 2003.

BIBLIOGRAPHY

[55] X. He and L. Deng, *Discriminative Learning for Speech Recognition*. Morgan & Claypool publishers, 2008.

[56] G. Heigold, H. Ney, R. Schlüter, and S. Wiesler, "Discriminative training for automatic speech recognition: Modeling, criteria, optimization, implementation, and performance," *IEEE Signal Processing Magazine*, vol. 29(6), 2012.

[57] L. Deng and X. Li, "Machine learning paradigms for speech recognition: An overview," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 21(5), 2013.

[58] R. McDonald, K. Crammer, and F. Pereira, "Online large-margin training of dependency parsers," in *Proceedings of the 43rd Annual Meeting of the ACL*, 2005.

[59] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[60] S. Shalev-Schwartz, "Online learning: Theory, algorithms, and applications," Ph.D. dissertation, Hebrew University, 2007.

[61] A. Rakhlin, "Lecture notes on online learning," 2009.

[62] S. Bubeck, "Introduction to online optimization," *Lecture Notes*, 2011.

[63] E. Hazan, *Optimization for Machine Learning*. MIT Press, 2011, ch. The Convex Optimization Approach to Regret Minimization.

BIBLIOGRAPHY

[64] S. Shalev-Schwartz, "Online learning and online convex optimization," *Foundations and Trends in Machine Learning*, vol. 4, 2012.

[65] P. Liang, "Statistical learning theory," *Lecture Notes*, 2014.

[66] E. Hazan, *Introduction to Online Convex Optimization*, 2014.

[67] V. N. Vapnik, *Statistical Learning Theory*. John Wiley & Sons, 1998.

[68] O. Dekel, J. Ding, T. Koren, and Y. Peres, "Online learning with composite loss functions," in *Proceedings of COLT*, 2014.

[69] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *Foundations and Trends in Machine Learning*, vol. 5, 2012.

[70] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proceedings of ICML*, 2003.

[71] A. Nemirovski and D. B. Yudin, *Problem Complexity and Method Efficiency in Optimization*. New York: Wiley, 1983.

[72] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2008.

[73] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Machine Learning*, 2007.

BIBLIOGRAPHY

[74] B. Widrow, "A statistical theory of adaptation," in *Proceedings of Adaptive Control Systems Symposium*, 1963.

[75] L. Bottou, *Online Learning and Neural Networks*. Cambridge University Press, 1998, ch. Online Algorithms and Stochastic Approximations.

[76] Shalev-Schwartz, Y. Singer, and N. Srebro, "Pegasos: Primal estimated sub-gradient solver for svm," 2007.

[77] M. K. Warmuth and A. K. Jagota, "Continuous and discrete-time nonlinear gradient descent: Relative loss bounds and convergence," in *Proceedings of 5th International Symposium on Artificial Intelligence and Mathematics*, 1998.

[78] K. S. Azoury and M. K. Warmuth, "Relative loss bounds for on-line density estimation with the exponential family of distributions," *Machine Learning*, vol. 43, 2001.

[79] A. Beck and M. Teboulle, "Mirror descent and nonlinear projected subgradient methods for convex optimization," *Operations Research Letters*, vol. 3, 2003.

[80] N. Srebro, K. Sridharan, and A. Tewari, "On the universality of online mirror descent," in *Proceedings of NIPS*, 2011.

[81] N. Parikh, "Proximal algorithms," *Foundations and Trends in Optimization*, 2013.

[82] J. Kivinen, "Exponentiated gradient versus gradient descent for linear predictors," *Information and Computation*, 1997.

BIBLIOGRAPHY

[83] R. E. Schapire and Y. Freund, *Boosting: Foundations and Algorithms*. MIT Press, 2012.

[84] J. Langford, L. Li, and T. Zhang, "Sparse online learning via truncated gradient," *Journal of Machine Learning and Research*, 2009.

[85] S. Shalev-Shwartz and A. Tewari, "Stochastic methods for l1 regularized loss minimization," in *Proceedings of ICML*, 2009.

[86] N. Littlestone, "Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm," *Machine Learning*, 1988.

[87] J. Kivinen and M. K. Warmuth, "The perceptron algorithm vs. winnow: Linear vs. logarithmic mistake bounds when few input variables are relevant," *Artificial Intelligence*, 1997.

[88] C. Gentile, "The robustness of the p-norm algorithms," *Machine Learning*, vol. 53, 2003.

[89] A. Grove, N. Littlestone, and D. Schuurmans, "General convergence results for linear discriminant updates," *Machine Learning*, vol. 43, 2001.

[90] Y. Nesterov, "Primal-dual subgradient methods for convex problems," *Mathematical Programming*, 2009.

[91] L. Xiao, "Dual averaging method for regularized stochastic learning and online optimization," in *Proceedings of NIPS*, 2009.

BIBLIOGRAPHY

[92] H. B. McMahan, "A unified view of regularized dual averaging and mirror descent with implicit updates," *arXiv:1009.3240*, 2011.

[93] N. Littlestone and M. Warmuth, "The weighted majority algorithm," *Journal of Information and Computation*, 1994.

[94] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, 1997.

[95] N. Cesa-Bianchi, Y. Mansour, and G. Stoltz, "Improved second-order bounds for prediction with expert advice," *Machine Learning*, vol. 66, 2007.

[96] H. B. McMahan, "Analysis techniques for adaptive online learning," *arXiv:1403.3465*, 2014.

[97] M. Ito and M. Fukuda, "A family of subgradient-based methods for convex optimization problems in a unifying framework," *arXiv:1403.6526*, 2014.

[98] J. Duchi and Y. Singer, "Effcient online and batch learning using forward backward splitting," *Journal of Machine Learning Research*, vol. 10, 2009.

[99] J. Duchi, S. Shalev-Schwartz, Y. Singer, and A. Tewari, "Composite objective mirror descent," in *Proceedings of COLT*, 2010.

[100] B. Kulis and P. L. Bartlett, "Implicit online learning," in *Proceedings of ICML*, 2010.

BIBLIOGRAPHY

[101] P. L. Bartlett and E. Hazan, "Adaptive online gradient descent," in *Proceedings of NIPS*, 2007.

[102] F. Orabona, K. Crammer, and N. Cesa-Bianchi, "A generalized online mirror descent with applications to classification and regression," *arXiv preprint arXiv:1304.2994*, 2013.

[103] F. Orabona, "Dimension-free exponentiated gradient," in *Proceedings of NIPS*, 2013.

[104] T. van Erven, P. Grünwald, W. M. Koolen, and S. de Rooij, "Adaptive hedge," in *Proceedings of NIPS*, 2011.

[105] F. Orabona and D. Pal, "Scale-free algorithms for online linear optimization," *arXiv:1502.05744*, 2015.

[106] N. Cesa-Bianchi, A. Conconi, and C. Gentile, "A second-order perceptron algorithm," *SIAM Journal on Computing*, vol. 34, 2005.

[107] M. Dredze, K. Crammer, and F. Pereira, "Online confidence-weighted learning," in *Proceedings of ICML*, 2008.

[108] F. Orabona and K. Crammer, "New adaptive algorithms for online classification," in *Proceedings of NIPS*, 2010.

[109] L. Yang, R. Jing, and J. Ye, "Online learning by ellipsoid method," in *Proceedings of ICML*, 2009.

BIBLIOGRAPHY

[110] S. Kakade, S. Shalev-Schwartz, and A. Tewari, "Regularization techniques for learning with matrices," *Journal of Machine Learning Research*, 2012.

[111] C. Rao, "Information and accuracy attainable in the estimation of statistical parameters," *Bulletin of the Calcutta Mathematical Society*, vol. 37, 1945.

[112] B. Efron, "Defining the curvature of a statistical problem (with applications to second order efficiency)," *The Annals of Statistics*, vol. 3, 1975.

[113] A. P. Dawid, "Discussion of efrons paper," *The Annals of Statistics*, vol. 3, 1975.

[114] S. Eguchi, "Second order efficiency of minimum contrast estimator in a curved exponential family," *The Annals of Statistics*, vol. 11, 1983.

[115] S. Amari, *Differential Geometrical Methods in Statistics*. Springer-Verlag, 1985.

[116] S. Amari and H. Nagaoka, *Methods of Information Geometry*. Oxford University Press, 2000.

[117] S. Amari, "Information geometry and its applications: Convex function and du-ally flat manifold," *Emerging Trends in Visual Computing, Lecture Notes in Computer Science, Springer-Verlag*, vol. 5416, 2008.

[118] S. Amari and A. Cichocki, "Information geometry of divergence functions," *Bulletin of the Polish Academy of Sciences Technical Sciences*, vol. 58, 2010.

BIBLIOGRAPHY

[119] J. Zhang and H. Matsuzoe, *Advances in Applied Mathematics and Global Optimization*. Springer, 2009, ch. Dualistic Riemannian Manifold Structure Induced from Convex Functions.

[120] F. Nielsen, "Pattern learning and recognition on statistical manifolds: An information-geometric review," *SIMBAD, Lecture Notes in Computer Science, Springer-Verlag*, vol. 7953, 2013.

[121] O. Calin and C. Udriste, *Geometric Modeling in Probability and Statistics*. Springer, 2014.

[122] I. Csiszár, "Eine informationstheoretische ungleichung und ihre anwendung auf den beweis der ergodizitat von markoffschen ketten," *Publications of the Mathematical Institute of the Hungarian Academy of Science*, 1963.

[123] N. N. Chentsov, "Statistical decision rules and optimal inference," *American Mathematical Society*, 1982.

[124] L. M. Bregman, "The relaxation method of finding the common points of convex sets and its application to the solution of problems in convex programming," *USSR Computational Mathematics and Mathematical Physics*, 1967.

[125] S. Amari, "α-divergence is unique, belonging to both f-divergence and bregman divergence classes," *IEEE Transactions on Information Theory*, vol. 55, 2009.

BIBLIOGRAPHY

[126] F. Nielsen and V. Garcia, "Statistical exponential families: A digest with flash cards," *arXiv:0911.4863*, 2011.

[127] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends in Machine Learning*, vol. 1, 2008.

[128] S. Amari, "Natural gradient descent works efficiently in learning," *Neural Computation*, vol. 10, 1998.

[129] N. L. Roux, P.-A. Manzagol, and Y. Bengio, "Topmoumoute online natural gradient algorithm," in *Proceedings of NIPS*, 2007.

[130] A. Honkela, M. Tornio, T. Raiko, and J. Karhunen, "Natural conjugate gradient in variational inference," *Lecture Notes in Computer Science, Springer-Verlag*, vol. 4985, 2008.

[131] R. Pascanu and Y. Bengio, "Revisiting natural gradient for deep networks," *arXiv preprint arXiv:1301.3584*, 2013.

[132] M. D. Hoffman, D. M. Blei, C. Wang, and J. Pasley, "Stochastic variational inference," *Journal of Machine Learning Research*, vol. 14, 2013.

[133] S. Arora, E. Hazan, and S. Kale, "The multiplicative weights update method: A meta-algorithm and applications," *Theory of Computing*, vol. 8, 2012.

[134] J. Steinhardt and P. Liang, "Adaptivity and optimism: An improved exponentiated gradient algorithm," in *Proceedings of ICML*, 2014.

BIBLIOGRAPHY

[135] A. Blum and Y. Monsour, *Algorithmic Game Theory*, 2007, ch. Learning, Regret Minimization, and Equilibria.

[136] Y. Freund and R. E. Schapire, "Adaptive game playing with multiplicative weights," *Games and Economic Behavior*, vol. 29, 1999.

[137] T. Zhang, "Regularized winnow methods," in *Proceedings of NIPS*, 2001.

[138] M. Collins, A. Globerson, T. Koo, X. Carreras, and P. L. Bartlett, "Exponentiated gradient algorithms for conditional random fields and max-margin markov networks," *Journal of Machine Learning Research*, vol. 9, 2008.

[139] S. Kale, "Efficient algorithms using the multiplicative weights update method," Ph.D. dissertation, Princeton University, 2007.

[140] K. Chaudhuri, Y. Freund, and D. Hsu, "A parameter-free hedging algorithm," in *Proceedings of NIPS*, 2009.

[141] M. Baes and M. Bürgisser, "Hedge algorithm and dual averaging schemes," *arXiv:1112.1275*, 2011.

[142] S. de Rooij, T. van Erven, P. Grünwald, and W. M. Koolen, "Follow the leader if you can, hedge if you must," *Journal of Machine Learning Research*, vol. 14(15), 2014.

[143] E. Hazan and T. Koren, "The computational power of optimization in online learning," *arXiv:1504.02089*, 2015.

BIBLIOGRAPHY

[144] P. P. B. Eggermont, "Multiplicative iterative algorithms for convex programming," *Linear Algebra and Its Applications*, vol. 130, 1990.

[145] A. N. Iusem, "An interior point multiplicative method for optimization under positivity constraints," *Acta Applicandae Mathematicae*, vol. 38, 1995.

[146] G. Raskutti and S. Mukherjee, "The information geometry of mirror descent," *arXiv:1310.7780*, 2013.

[147] Y. Freund and R. E. Shapire, "Large margin classification using the perceptron algorithm," *Machine Learning*, vol. 37(3), 1999.

[148] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," in *Proceedings of HLT-NAACL*, 2010.

[149] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics, 2009.

[150] A. Ittycheria and S. Roukos, "Direct translation model 2," in *Proceedings of NAACL*, 2007.

[151] M. Snover, B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul, "A study of translation edit rate with targeted human annotation," in *Proceedings of the AMTA*, 2006.

[152] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, , and

BIBLIOGRAPHY

E. Herbst, "Moses: Open source toolkit for statistical machine translation," in *Proceedings of ACL*, 2007.

[153] N. Durrani, B. Haddow, K. Heafield, , and P. Koehn, "Edinburghs machine translation systems for european language pairs," in *Proceedings of the Eighth Workshop on Statistical Machine Translation (WMT)*, 2013.

[154] C. Tsallis, "Possible generalization of boltzmanngibbs statistics," *Journal of Statistical Physics*, vol. 52, 1988.

[155] ——, *Introduction to Nonextensive Statistical Mechanics*. Springer, 2009.

[156] ——, "What are the numbers that experiments provide?" *Quimica Nova*, vol. 17, 1994.

[157] T. Yamano, "Some properties of q-logarithm and q-exponential functions in tsallis statistics," *Physics A*, vol. 305, 2002.

[158] H. R. Quiceno, G. I. Loaiza, and J. C. Arango, *Geometric Theory of Information*. Springer, 2014, ch. A Riemannian Geometry in the q-Exponential Banach Manifold Induced by q-Divergences.

[159] S. Amari and A. Ohara, "Geometry of q -exponential family of probability distributions," *Entropy*, vol. 13, 2011.

[160] N. Ding, "Statistical machine learning in the t-exponential family of distributions," Ph.D. dissertation, Purdue University, 2013.

BIBLIOGRAPHY

[161] J. Naudts, "Deformed exponentials and logarithms in generalized thermostatistics," *Physics A*, vol. 316, 2002.

[162] ——, "Estimators, escort probabilties, and φ-exponential families in statistical physics," *Journal of Inequalities in Pure and Applied Mathematics*, vol. 5, 2004.

[163] S. Furuichi, K.Yanagi, and K.Kuriyama, "Fundamental properties of tsallis relative entropy," *arXiv:cond-mat/0406178*, 2005.

[164] M. Teboulle, "Entropic proximal mappings with applications to nonlinear programming," *Mathematics of Operations Research*, vol. 17(3), 1992.

[165] ——, "Convergence of proximal-like algorithms," *SIAM Journal of Optimization*, vol. 7(4), 1997.

[166] A. N. Iusem, B. F. Svaiter, and M. Teboulle, "Entropy-like proximal methods in convex programming," *Mathematics of Operations Research*, vol. 19(4), 1994.

[167] ——, "Multiplicative interior gradient methods for minimization over the nonnegative orthant," *SIAM Journal of Control and Optimization*, vol. 34, 1996.

[168] R. E. Mahony and R. C. Williamson, "Prior knowledge and preferential structures in gradient descent learning algorithms," *Journal of Machine Learning Research*, 2001.

[169] ——, "Riemannian structure of some new gradient descent learning algorithms," in

IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium, 2000.

[170] K. Krakowski, R. Mahony, R. Williamson, and M. Warmuth, "Online learning on spheres," in *(Unpublished)*, 2005.

[171] K. A. Krakowski, R. E. Mahony, R. C. Williamson, and M. K. Warmuth, "A geometric view of non-linear on-line stochastic gradient descent," *(Unpublished)*, 2008.

[172] R. E. Mahony, K. Blackmore, K. A. Krakowski, and R. C. Williamson, "Intrinsic geometry of stochastic gradient descent algorithms," in *(Unpublished)*, 2006.

[173] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers, 2004.

[174] ——, "A method of solving a convex programming problem with convergence rate o(1/k2)," *Doklady AN SSSR*, vol. 269(3), 1983.

[175] ——, "On an approach to the construction of optimal methods of minimization of smooth convex function," *Èkonom. i. Mat. Metody*, vol. 24, 1988.

[176] ——, "Smooth minimization of non-smooth functions," *Mathematical Programming*, vol. 103, 2005.

[177] ——, "Gradient methods for minimizing composite objective function," *Center for Operations Research and Econometrics (CORE) Discussion Papers*, 2007.

BIBLIOGRAPHY

[178] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal of Imaging Sciences*, vol. 2(1), 2009.

[179] L. Györfi and T. Nemetz, "f-dissimilarity: A generalization of the affinity of several distributions," *Annals of the Institute of Statistical Mathematics*, vol. 30(1), 1978.

[180] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of ICML*, 2001.

[181] A. McCallum and W. Li, "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons," in *Proceedings of the Seventh Conference on Natural Language Learning at NAACL-HLT*, 2003.

[182] F. Pent, F. Feng, and A. McCallum, "Chinese segmentation and new word detection using conditional random fields," in *Proceedings of the International Conference on Computational Linguistics (COLING)*, 2004.

[183] D. C. Liu and J. Nocedal, "On the limited memory method for large scale optimization," *Mathematical Programming*, 1989.

[184] N. N. Schraudolph, "Local gain adaptation in stochastic gradient descent," in *Proceedings of International Conference on Artificial Neural Networks (ICANN)*, 1999.

[185] S. N. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. P. Murphy, "Accelerated training of conditional random fields with stochastic gradient methods," 1999.

BIBLIOGRAPHY

[186] A. Bordes, L. Bottou, and P. Gallinari, "Careful quasi-newton stochastic gradient descent," *Journal of Machine Learning Research*, 2009.

[187] L. Bottou, *Neural Networks: Tricks of the Trade, Second Edition*. Lecture Notes in Computer Science, 2012, ch. Stochastic Gradient Descent Tricks.

[188] N. Okazaki, "Crfsuite: A fast implementation of conditional random fields (crfs)," 2007. [Online]. Available: http://www.chokkan.org/software/crfsuite

[189] E. F. T. K. Sang and S. Buchholz, "Introduction to the conll-2000 shared task: Chunking," in *Proceedings of Conference on Computational Natural Language Learning (CoNLL)*, 2000.

[190] T. Hazan, S. Polak, and A. Shashua, "Sparse image coding using a 3d non-negative tensor factorization," 2005.

[191] A. Shashua and T. Hazan, "Non-negative tensor factorization with applications to statistics and computer vision," in *Proceedings of the ICML*, 2005.

[192] D. Cai, X. He, and J. Han, "Tensor space model for document analysis," 2006.

[193] S. Cohen and M. Collins, "Tensor decomposition for fast parsing with latent-variable pcfgs," in *Proceedings of NIPS*, 2012.

[194] S. Cohen and G. Satta, "Approximate pcfg parsing using tensor decomposition," in *Proceedings of NAACL-HLT*, 2013.

BIBLIOGRAPHY

[195] T. V. de Cruys Thierry Poibeau and A. Korhonen, "A tensor-based factorization model of semantic compositionality," in *Proceedings of NAACL-HLT*, 2013.

[196] T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Review*, 2009.

[197] M. Fazel, "Matrix rank minimization with applications," Ph.D. dissertation, Stanford University, 2002.

[198] D. Cai, X. He, and J. Han, "Learning with tensor representation," Department of Computer Science, University of Illinois at Urbana-Champaign, Tech. Rep., 2006.

[199] R. T. Rockafellar, *Convex Analysis*. Princeton, NJ: Princeton University Press, 1972.

[200] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[201] D. P. Bertsekas, *Convex Optimization Theory*. Nashua, NH: Athena Scientific, 2014.

[202] F. Nielsen, "Legendre transformation and information geometry," Tech. Rep. CIG-MEMO2, 2010, http://www.informationgeometry.org.

[203] W. M. Boothby, *An Introduction to Differentiable Manifolds and Riemannian Geometry*. Orlando, FL: Academic Press, 1986.

BIBLIOGRAPHY

[204] J. M. Lee, *Riemannian Manifolds: An Introduction to Curvature*. New York: Springer-Verlag, 1997.

[205] A. McInerney, *First Steps in Differential Geometry: Riemannian, Contact, Symplectic*. New York: Springer, 2013.

[206] L. Pardo, *Statistical Inference Based on Divergence Measures*. Boca Raton, FL: Chapman and Hall/CRC, 2005, ch. Divergence Measures: Definition and Properties.

[207] A. Cichocki and S. Amari, "Families of alpha- beta- and gamma- divergences: Flexible and robust measures of similarities," *Entropy*, vol. 12, 2010.

[208] M. Basseville, "Divergence measures for statistical data processing - an annotated bibliography," *Signal Processing*, vol. 93, 2013.

[209] J. Zhang, "Divergence function, duality, and convex analysis," *Neural Computation*, vol. 16, 2004.

[210] S. M. Ali and S. D. Silvey, "A general class of coefficients of divergence of one distribution from another," *Journal of Royal Statistics Society*, vol. 28, 1966.

[211] Y. Qiao and N. Minematsu, "A study on invariance of f-divergence and its applications to speech recognition," *IEEE Transactions on Speech Processing*, vol. 58(7), 2010.

[212] F. Liese and I. Vajda, "On divergences and informations in statistics and information theory," *IEEE Transactions on Information Theory*, vol. 52, 2006.

BIBLIOGRAPHY

[213] I. Sason, "Bounds on f-divergences and related distances," *arXiv:1403.7164*, 2014.

[214] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *The Annals of Mathematical Statistics*, vol. 23, 1952.

[215] N. Cressie and T. R. C. Reed, "Multinomial goodness-of-fit tests," *Journal of Royal Statistical Society, Series B*, vol. 46, 1984.

[216] B. Póczos and J. Schneider, "On the estimation ofα-divergences," in *Proceedings of AISTATS*, 2011.

[217] F. Nielsen and R. Nock, "On rényi and tsallis entropies and divergences for exponential families," *arXiv:1105.3259*, 1984.

[218] A. Rényi, "On measures of information and entropy," in *Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability*, 1961.

[219] T. van Erven and P. Harremoes, "Rényi divergence and kullbackleibler divergence," *IEEE Transactions on Information Theory*, vol. 60, 2014.

[220] K. Tsuda, G. Rätsch, and M. K. Warmuth, "Matrix exponentiated gradient updates for on-line learning and bregman projection," *Journal of Machine Learning Research*, vol. 6, 2005.

[221] I. S. Dhillon and J. A. Tropp, "Matrix nearness problem with bregman divergences," *SIAM Journal of Matrix Analysis and Applications*, vol. 29(4), 2007.

BIBLIOGRAPHY

[222] B. Kulis, M. A. Sustik, and I. S. Dhillon, "Low-rank kernel learning with bregman matrix divergences," *Journal of Machine Learning Research*, vol. 10, 2009.

[223] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh, "Clustering with bregman divergences," *Journal of Machine Learning Research*, vol. 6, 2005.